VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Programová implementace USB-RS232 převodníku v procesoru Microchip

Software Implementation of USB-RS232 Converter

2012

Bc. Marek Darmo, DiS.

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Marek Darmo, DiS.**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Programová implementace USB-RS232 převodníku v procesoru Microchip
Software Implementation of USB-RS232 Converter

Zásady pro vypracování:

Navrhněte programovou emulaci převodníku USB-RS232 firmy FTDI v mikroprocesoru PIC18Fx. Vyberte procesor s fyzicky implementovaným USB rozhraním a s použitím knihoven výrobce proveďte potřebnou implementaci kódu v jazyce C.

1. Porovnejte mikropočítače Microchip s USB rozhraním a vyberte vhodný typ.
2. Technický a programátorský popis převodníku USB-RS232 firmy FTDI.
3. Návrh programového řešení s použitím dostupných USB implementací.
4. Zvažte možnost implementace dvou a více převodníků jedním mikropočítačem.
5. Ověření kompatibility ve více OS, testování stability, spolehlivosti a propustnosti.

Seznam doporučené odborné literatury:

Kainka, Burkhard: USB - Měření, řízení a regulace pomocí sběrnice USB, BEN - technická literatura, ISBN 80-7300-073-3
http://www.microchip.com - technické listy
http://www.ftdichip.com

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Petr Olivka**

Datum zadání: 18.11.2011
Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 4.května 2012

Marek Domo

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli a svým dílem tak přispěli k celkové kvalitě práce. Děkuji Ing. Stanislavu Darmovi, mému otci, za pečlivé korektury a připomínky ke znění textu. Děkuji Petru Hudečkovi, mému kamarádovi, který mi umožnil otestovat emulátor na jeho osobním notebooku MacBook Pro s operačním systémem Mac OS X.

Zvláště pak děkuji panu Ing. Petru Olivkovi, vedoucímu mé diplomové práce, za jeho trpělivost, cenné rady a podnětné připomínky. Zvláště oceňuji jeho rychlou odezvu při emailové komunikaci, která mi mnohdy pomohla při mých časových možnostech kombinovaného studia.

Abstrakt

Cílem práce je navrhnout programovou emulaci převodníku USB-RS232 firmy FTDI v mikroprocesoru řady PIC18 výrobce Microchip s použitím knihoven výrobce v jazyce C.

Klíčová slova: Microchip, PIC18, FTDI, USB, RS-232, USB-RS232, diplomová práce

Abstract

The goal of this thesis is to design a program emulation of FTDI USB-RS232 convertor in the PIC18 series microprocessor of Microchip manufacturer using the manufacturer's C library.

Keywords: Microchip, PIC18, FTDI, USB, RS-232, USB-RS232, diplomová práce

Seznam použitých zkratek a symbolů

USB	–	Universal Serial Bus
RS-232	–	Standart pro sériovou komunikaci
FTDI	–	Future Technology Devices International Ltd. ©
UART	–	Universal Asynchronous Receiver/Transmitter
DCE	–	Data Communications Equipment (modem)
DTE	–	Data Terminal Equipment (computer, printer)
CTS	–	Clear To Send [DCE -to- DTE]
DCD	–	Data Carrier Detected (Tone from a modem) [DCE -to- DTE]
DSR	–	Data Set Ready [DCE -to- DTE]
DTR	–	Data Terminal Ready [DTE -to- DCE]
RTS	–	Request To Send [DTE -to- DCE]
RI	–	Ring Indicator (ringing tone detected)
RxD	–	Received Data [DCE -to- DTE]
TxD	–	Transmitted Data [DTE -to- DCE]
PIM	–	Processor Interface Module - (Microchip hw Plug-In Module)
XLP	–	eXtreme Low Power
WDT	–	Watch Dog Timer - pro automatický reset mikroprocesoru
RTC	–	Real Time Clock - pracovní frekvence mikroprocesoru
ICSP™	–	In-Circuit Serial Programming™
ICD	–	In-Circuit Debug
MIPS	–	Million Instructions Per Second
PLL	–	Phase Locked Loop
RTCC	–	Real Time Clock Counter
CTMU	–	Charge Time Measurement Unit
I/O	–	Vstupně/Výstupní
BRG	–	Baud Rate Generator
SIE	–	USB Serial Interface Engine
SIO	–	Serial Input Output module
CDC	–	USB Specifikace Communication Device Class
VID	–	Vendor ID
PID	–	Product ID
EUSART	–	Enhanced Universal Synchronous Asynchronous Receiver Transmitter

API	–	Application Programming Interface
PIC18	–	Vývojová řada mikroprocesorů od výrobce Microchip
OS	–	Operační systém
I^2C	–	Inter-Integrated Circuit

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

V dnešní době stále ještě existuje velké množství periférií založených na komunikaci po sériové sběrnici RS232 pro její dostupnost, jednoduchost a univerzálnost. Porty RS232 z počítačových systémů ustupují a v dnešní době se v podstatě u nových počítačových systémů nevyskytují. Jsou nahrazovány dokonalejším nástupcem, porty a sběrnicí Universal Serial Bus (USB). Nicméně pro svoji jednoduchost a univerzálnost jsou porty RS232 stále používány v malých „embedded" zařízeních, což jsou mikroprocesorem řízené aplikace, které jsou jedním ze svých portů vybavovány právě sériovou komunikací po sběrnici RS232. A zde nastává otázka. Jak připojit takovéto zařízení k vyššímu, modernímu nadřazenému systému, který není vybaven porty RS232?

Jednou z odpovědí na otázku je použít převodník mezi sběrnicí RS232 a sběrnicí USB. Tohoto si je vědoma i společnost FTDI pocházející z Velké Británie, která nabízí speciální řešení na převod periférií na sběrnici USB, kterou jsou všechny novodobé počítačové systémy bez výhrady vybaveny. Společnost tedy nabízí i převodníky RS232 na USB a dodává také ke svým převodníkům volné ovladače pro širokou škálu operačních systémů.

Nicméně FTDI USB-RS232 převodník je externí hardwarovou součástkou, která tvoří most mezi vyšším systémem vybaveným USB sběrnicí, například osobním počítačem, a nižším systémem například mikroprocesorem řízeným „embedded" zařízením. V současnosti však výrobci mikroprocesorů začínají integrovat USB zařízení přímo do svých hardwarových součástek. To zřejmě vedlo zadavatele této práce k myšlence vyzkoušet emulaci FTDI zařízení v mikroprocesoru vybaveným USB sběrnicí tak, aby bylo možno využít volně dostupných ovladačů, které dodává společnost FTDI. Takto lze velice rychle odladit spojení vyššího a prototypu nižšího systému přes sběrnici USB, které bude následně v reálné aplikaci nahrazeno přímo obvodem FTDI nebo napsáním vlastních ovladačů ve vyšším systému, neboť finální firmware v mikroprocesoru nižšího systému bude již úspěšně implementován.

Tato práce čtenáře postupně provede informacemi o integrovaných obvodech použitých pro emulaci a základy originálního převodníku společnosti FTDI. Dále bude popsána samotná programová implementace emulátoru a probrána možnost implementace dvou a více převodníků jedním mikroprocesorem. Nakonec implementované zařízení řádně otestujeme.

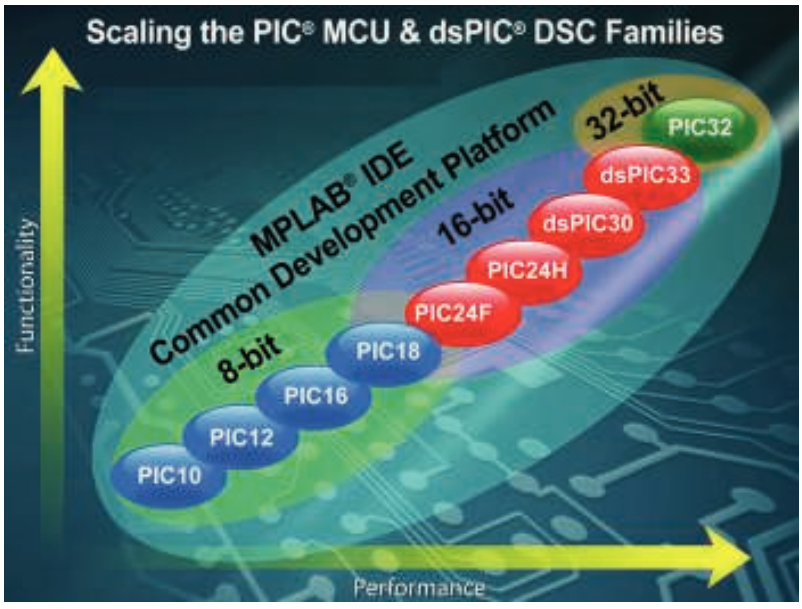
2 Výběr mikroprocesoru pro emulaci

2.1 Mikroprocesory od výrobce Microchip

Jedním z požadavků zadání diplomové práce je výběr mikroprocesoru, ve kterém bude emulátor aplikován a vyzkoušen. Návrhem je emulovat převodník USB-RS232 firmy FTDI v mikroprocesoru řady PIC18 od výrobce Microchip. Výrobce Microchip používá ve své produktové řadě hned několik mikroprocesorů vybavených rozhraním USB.

2.2 Mikroprocesory PIC18 s rozhraním USB

Výběr řady mikroprocesorů PIC18 pro aplikaci emulátoru převodníku USB-RS232 nebyl náhodný. V době zadání byla řada mikroprocesorů PIC18 nejnižší řadou, která byla rozhraním USB vybavena. Obrázek 1 *„Produktová řada mikroprocesorů výrobce Microchip, zdroj: [1]“* na straně 11 ukazuje postavení řady PIC18 v celé škále dostupných mikroprocesorů výrobce Microchip. Vidíme, že řada PIC18 je ve středu všech nabízených produktů a na rozhraní architektury 8-mi a 16-ti bitů.



Obrázek 1: Produktová řada mikroprocesorů výrobce Microchip, zdroj: [1]

2.3 Mikroprocesory PIC16 s rozhraním USB

V počátcích realizace této práce byla řada PIC18 nejnižší řadou mikroprocesorů, ve které bylo výrobcem Microchip USB rozhraní implementováno. Nyní však na konci realizace této práce výrobce Microchip implementoval USB rozhraní i do své nižší řady PIC16 do nově připravovaných mikroprocesorů. Připravované mikroprocesory jsou celkem čtyři jmenovitě PIC16F1454, PIC16F1455 a PIC16F1458, PIC16F1459. K březnu 2012 je výrobní status jmenovaných mikroprocesorů „Future Product“ což znamená, že jsou ve stavu přípravy a budou dostupny distributorům v nejbližších týdnech.

2.4 Výběr vhodného typu mikroprocesoru PIC18

Pro výběr vhodného mikroprocesoru lze s výhodou použít nástroj pro parametrický výběr součástek přímo na webové stránce výrobce a použít jej k vyhledání mikroprocesorů s rozhraním USB. Kriteriem zadání pro vyhledávání byla tedy produktová řada PIC18 s rozhraním USB. Vyhledávací nástroj dle těchto dvou kriterií nabídl seznam celkem třiceti mikroprocesorů. Protože všechny mikroprocesory v seznamu splňují specifikaci Full Speed USB 2.0, jako následující kriterium užšího výběru jsem zvolil novější typy řady „J“ a současně existenci vývojového kitu od výrobce Microchip pro vybraný typ mikroprocesoru. Po podrobném prohlédnutí dostupných vývojových kitů výrobce Microchip, které jsou primárně nebo doplňkově určeny pro testování a vývoj aplikací pro rozhraní USB, jsem vybral mikroprocesor řady PIC18 typ PIC18F46J50, pro který existuje cenově dostupný vývojový kit „Microchip PIC18F46J50 FS USB Demo Board Part Number: MA180024“.

2.5 Popis mikroprocesoru PIC18F46J50

Mikroprocesor Microchip PIC18F46J50 je 44 pinový, nízkopříkonový, vysoce výkoný USB mikroprocesor s nanoWatt XLP technologií. Zkratka technologie XLP je eXtreme Low Power v překladu extrémně nízký příkon. Tato technologie se vyznačuje nízkou spotřebou v režimu spánku mikroprocesoru 20nA nebo méně, nízkou spotřebou WDT časovače pro automatický reset mikroprocesoru 400nA nebo méně a nízkou spotřebou RTC pracovní frekvence mikroprocesoru 500nA nebo méně.

2.5.1 Všeobecné vlastnosti a parametry mikroprocesoru

V této kapitole si představíme některé užitečné vlastnosti mikroprocesoru PIC18F46J50. Kompletní seznam vlastností lze najít v katalogovém listu [6] výrobce.

- 64K bajtů FLASH paměti a 3.8K bajtů SRAM

- nanoWatt XLP™ Technology ideální pro bateriově napájené aplikace
- nastavitelné priority přerušení
- mikroprocesor je samoprogramovatelný s externím řídícím softwarem
- ICSP™ (In-Circuit Serial Programming™)
- ICD (In-Circuit Debug)
- napájení v rozmezí od 2.0V do 3.6V, interní 2.5V regulátor
- dva externí režimy oscilátoru do 48 MHz (12 MIPS)
- nastavitelný nízkopříkonový interní RC oscilátor 31 kHz až 8 MHz nebo do 48 MHz s PLL a druhý oscilátor Timer1 @ 32 kHz

2.5.2 Vlastnosti mikroprocesoru spojené se sběrnicí USB

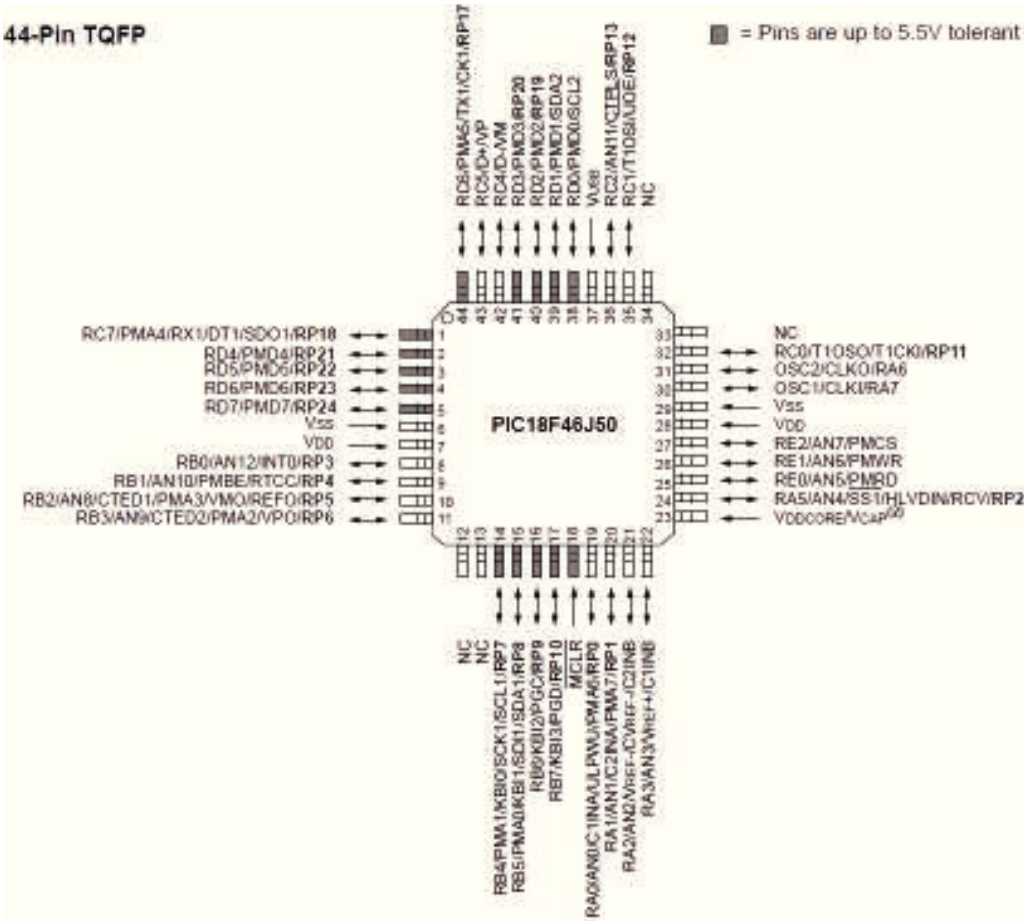
Níže jsou uvedeny vlastnosti mikroprocesoru spojené se sběrnicí USB.

- splňuje specifikaci USB V2.0
- podporuje Full Speed (12 Mbps) a Low Speed (1.5 Mbps)
- podporuje všechny transfery dat (Control, Interrupt, Isochronous a Bulk)
- podporuje až 32 Endpointů (16 obousměrných)
- integrovaný USB modul může využít jakoukoliv část RAM mikroprocesoru pro Buffery Endpointů
- vysoce přesný Interní Oscilátor ($\pm 0.15\%$ typ.) pro USB, integrovaný USB vysílač/-přijímač nepotřebuje externí oscilátor (krystal)

2.5.3 Ostatní vlastnosti a vybavení mikroprocesoru periferiemi

Mezi ostatní vlastnosti mikroprocesoru PIC18F46J50 řadíme zejména jeho periférie či moduly. Vyobrazení mikroprocesoru a rozložení jeho pinů je na obrázku 2 „*Rozložení pinů mikroprocesoru PIC18F46J50, zdroj: [6]*“ na straně 14.

- softwarové remapování pinů periférií užitečné pro flexibilní řešení
- harwarový RTCC poskytuje hodiny, kalendář a funkce alarmu



Obrázek 2: Rozložení pinů mikroprocesoru PIC18F46J50, zdroj: [6]

- integrovaná jednotka CTMU (Charge Time Measurement Unit) podporuje kapacitní dotykové panely
- 8 x 8 hardwarová dělička
- 2x modul ECCP - Enhanced Capture / Compare / PWM modules
- 2x modul MSSP sériových portů pro komunikaci SPI nebo I2C^{TM}
- 2x modul Enhanced USART
- 8-bit Paralelní port
- dvojité analogové komparátory
- 13-ti kanálový 10-bit ADC
- 5.5V tolerantní digitální vstupy

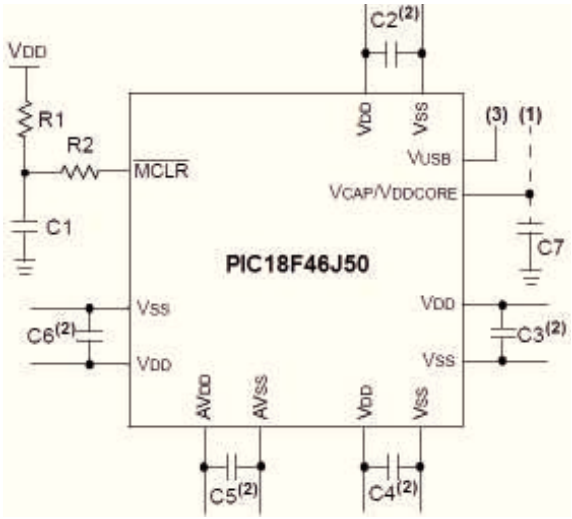
2.5.4 Funkční zapojení mikroprocesoru PIC18F46J50

Minimální funkční zapojení mikroprocesoru je na obrázku 3 „*Minimální funkční zapojení mikroprocesoru PIC18F46J50, zdroj: [6]*“ na straně 15. Vždy musí být zapojeny minimálně všechny napájecí piny VDD, VSS a všechny piny AVDD, AVSS a to i v případě, kdy analogové moduly mikroprocesoru nebudou použity. Na všech napájecích pinech musí být přítomen oddělovací kondenzátor o hodnotě alespoň 100nF/10V až 20V.

Dále musí být vždy zapojen pin ($\overline{MCLR}$) Master Clear sloužící ke správnému resetu mikroprocesoru nebo k jeho programování či uvedení do režimu ladění. Pokud programování nebo ladění nebude použito pin ($\overline{MCLR}$) se připojí přímo na napájecí pin VDD. Typické hodnoty součástek jsou

$$R_1 \leq 10k\Omega, R_2 \leq 470\Omega, C_1 \leq 100nF/10V \text{ až } 20V.$$

Pro stabilizaci interního napěťového regulátoru se musí vždy k pinu VCAP/VDDCORE připojit kondenzátor $10\mu F$ na zem.



Obrázek 3: Minimální funkční zapojení mikroprocesoru PIC18F46J50, zdroj: [6].

Následující popis zapojení pinů se týká případů, kdy je chceme v aplikaci použít.

Piny PGC/PGD se používají pro programování mikroprocesoru pomocí ICSP™ a pro účely ladění. Konkrétní zapojení těchto pinů se řídí jednotlivým použitým programátorem a je většinou popsáno v dokumentaci k samotnému programátoru například programátor PicKit3 v literatuře [14]. Pro výrobu vlastního programátoru lze základní informace o ICSP™ programování a harwarové specifikaci připojení pinů PGC/PGD získat v referenci [16].

Pokud je jako zdroj kmitočtu použit externí oscilátor musí být zapojen na piny OSCI a OSCO.

Pokud použijeme pro analogové moduly externí referenční napětí, přivádí se na piny VREF+/VREF-.

Nakonec všechny I/O piny by měly být nakonfigurovány jako výstupní a nastaveny do logicky nízké urovně. Případně lze na nepoužité I/O piny připojit rezistor o velikosti $1k\Omega$ až $10k\Omega$ k zemi a nastavit výstup do logicky nízké urovně.

2.5.5 Dostupné konfigurace oscilátoru

Mikroprocesor PIC18F46J50 má oproti celé řadě PIC18 přepracovaný modul oscilátoru pro řízení kmitočtu. Kromě již zmiňovaného přesného a stabilního vnitřního oscilátoru se vyznačuje modul oscilátoru také kompatibilitou pro USB obou rychlostí Low-Speed i High-Speed. Navíc má mikroprocesor PIC18F46J50 další děličku prescaller a postscaller což dále zvyšuje možnosti nastavení širokého rozsahu kmitočtů. Mikroprocesor PIC18F46J50 má dostupných celkem 8 režimů oscilátoru ze kterých si lze vybrat dle použití mikroprocesoru ten nejvhodnější. Dostupné jsou celkem dva režimy externího hodinového signálu pojmenované ECPLL a EC, hodinový kmitočet se přivádí na vstup RA6 mikroprocesoru. Další dva režimy slouží pro připojení externího krystalu na piny RA6 a RA7 mikroprocesoru a jsou pojmenovány HSPLL a HS. Nakonec poslední 4 režimy jsou spojeny s přesným interním oscilátorem a jejich režim je pojmenován INTOSCPLLO, INTOSCPLL, INTOSCO a INTOSC. Režimy ECPLL, EC, INTOSCPLLO a INTOSCO poskytují hodinový výstup (CLKO) na pinu RA6 mikroprocesoru o velikosti $\frac{1}{4}$ kmitočtu periférií obvodu.

Vzhledem ke zvláštním požadavkům na práci USB modulu je nutný také odlišný přístup k problematice hodinového kmitočtu v mikroprocesoru. Aby bylo možno modul USB provozovat na obou rychlostech Low-Speed i High-Speed je nutné zajistit v modulu oscilátoru obě frekvence jak 6MHz tak i 48MHz. Nakonec vnitřní frekvence mikroprocesoru může být zcela odlišná od frekvence pro sběrnici USB. Na všechny tyto zvláštnosti je modul oscilátoru mikroprocesoru PIC18F46J50 připraven a je vybaven modulem PLL s pevnou frekvencí 96MHz od které lze následně odvodit různé frekvence zvlášť pro USB a pro jádro mikroprocesoru. Všechny režimy modulu oscilátoru jsou podrobně popsány v katalogovém listu [6] výrobce. Konkrétní nastavení modulu oscilátoru pro práci s USB periférií si představíme v kapitole 4.5.3.4 „*Konfigurace v „main.c““* na straně 59.

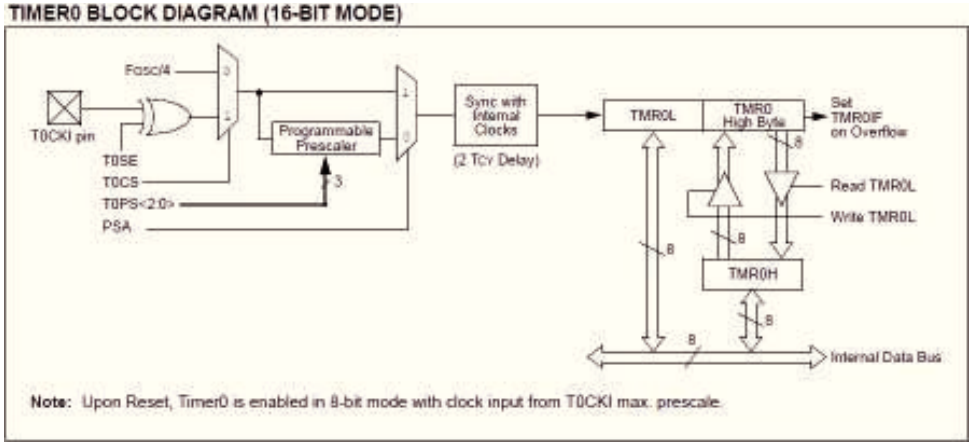
2.5.6 Popis použitých modulů mikroprocesoru

V této části si blíže popíšeme moduly mikroprocesoru PIC18F46J50, které byly bezprostředně použity při realizaci této práce a jsou tedy nutným teoretickým základem k pochopení dalšího textu. Na tyto teoretické části se budeme dále odvolávat a používat je.

2.5.6.1 TIMER0

Modul TIMER0 je jedním z časovačů, kterými je mikroprocesor PIC18F46J50 vybaven. TIMER0 může pracovat v režimu časovače nebo počítadla externích impulsů a to jak v 8-mi bitovém tak i v 16-ti bitovém režimu. Časovač je vybaven děličkou kmitočtu Prescaler, která je softwarově programovatelná. Přetečení časovače může být navázáno na přerušení mikroprocesoru. V 16-ti bitovém režimu dochází k přetečení z hodnoty FFFFh do 0000h.

Funkce časovače TIMER0 je nastavitelná pomocí registru T0CON. Od chvíle kdy je časovač povolen, je inkrementován s každým hodinovým cyklem pokud není nastavena dělička Prescaler. Potom se kmitočet dělí poměrem děličky, která je nastavitelná v rozmezí 1:2 až 1:256. Nastavení časovače se provádí zápisem do registru TMR0. V případě, že je použit časovač v 16-ti bitovém režimu, zapíše se nejdříve horní bajt časovače TMR0H, který je bufferován a následně až dolní bajt TMR0H. Současný zápis 16-ti bitové hodnoty do registru TMR0 provede mikroprocesor automaticky. Naopak čtení registru musí probíhat opačně, nejdříve se čte dolní registr TMR0L, mikroprocesor připraví do bufferu hodnotu horního bajtu TMR0H a pak ji lze bezpečně přečíst. Tímto postupem se dosahuje současné aktualizace celého registru TMR0.



Obrázek 4: 16-ti bitový časovač TIMER0 mikroprocesoru PIC18F46J50, zdroj: [6]

Vnitřní blokové schéma časovače TIMER0 pracujícího v 16-ti bitovém režimu je na obrázku 4 „*16-ti bitový časovač TIMER0 mikroprocesoru PIC18F46J50, zdroj: [6]*“ na straně

17. V emulátoru je tento časovač použit jako 16-ti bitový pro funkci Latency Timer, kterou si představíme v kapitole 3.4.6 „*Latency Timer*“ na straně 39.

2.5.6.2 EUSART

Mikroprocesor PIC18F46J50 je vybaven celkem dvěma nezávislými identickými moduly USART1 a USART2. Modul USART může pracovat ve full-duplex asynchronním režimu nebo half-duplex synchronním režimu. Zvláštním vybavením USART modulu jsou:

- Asynchronous Auto-wake-up on character reception
- Asynchronous Auto-baud calibration
- Asynchronous 12-bit Break character transmission
- Synchronous Master (half-duplex) with selectable clock polarity
- Synchronous Slave (half-duplex) with selectable clock polarity

Tyto zvláštnosti definuje výrobce Mikrochip jako rozšířený USART a nazývá jej označením EUSART.

Činnost každého EUSART modulu je řízena pomocí tří registrů:

- Transmit Status and Control (TXSTAx)
- Receive Status and Control (RCSTAx)
- Baud Rate Control (BAUDCONx)

Písmenko „x“ označuje číslo EUSART modulu buď 1 nebo 2.

BAUD RATE FORMULAS

Configuration Bits			BRG/EUSART Mode	Baud Rate Formula
SYNC	BRG16	BRGH		
0	0	0	8-bit/Asynchronous	Fosc/[64 (n + 1)]
0	0	1	8-bit/Asynchronous	Fosc/[16 (n + 1)]
0	1	0	16-bit/Asynchronous	
0	1	1	16-bit/Asynchronous	Fosc/[4 (n + 1)]
1	0	x	8-bit/Synchronous	
1	1	x	16-bit/Synchronous	

Legend: x = Don't care, n = value of SPBRGHx:SPBRGx register pair

Obrázek 5: Výpočet přenosové rychlosti mikroprocesoru PIC18F46J50, zdroj: [6]

Modul EUSART je v emulátoru použit jako komunikační interfejs pro sériovou komunikaci RS-232 po datové sběrnici. Konkrétně byl použit modul EUSART1 pracující ve full-duplex asynchronním režimu.

Další vlastnosti modulu EUSART nastaveného v asynchronním režimu jsou uvedeny v katalogovém listu [6] výrobce mikroprocesoru PIC18F46J50, zejména pak blokové schéma vysílače a přijímače a jejich specifických funkcí.

Mikroprocesor PIC18F46J50 má oddělený generátor přenosové rychlosti (BRG) Baud Rate Generator. Generátor BRG je 8-mi nebo 16-ti bitový a podporuje oba režimy činnosti modulu EUSART jak asynchronní tak synchronní. Šířka bitů generátoru BRG je programově volitelná v registru BAUDCONx. Přenosová rychlost generátoru BRG je nastavitelná v registrovém páru SPBRGHx:SPBRGx a v asynchronním režimu v registrech BRGH (TXSTAx<2>) a BRG16 (BAUDCONx<3>).

Výpočet přenosové rychlosti pro různé režimy modulu EUSART jsou uvedeny v tabulce na obrázku 5 „*Výpočet přenosové rychlosti mikroprocesoru PIC18F46J50, zdroj: [6]*“ na straně 19. Přesné nastavení pro účely našeho emulátoru probereme v sekci 4.5.8.2 „*Výpočet pro nastavení baudrate*“ na straně 75.

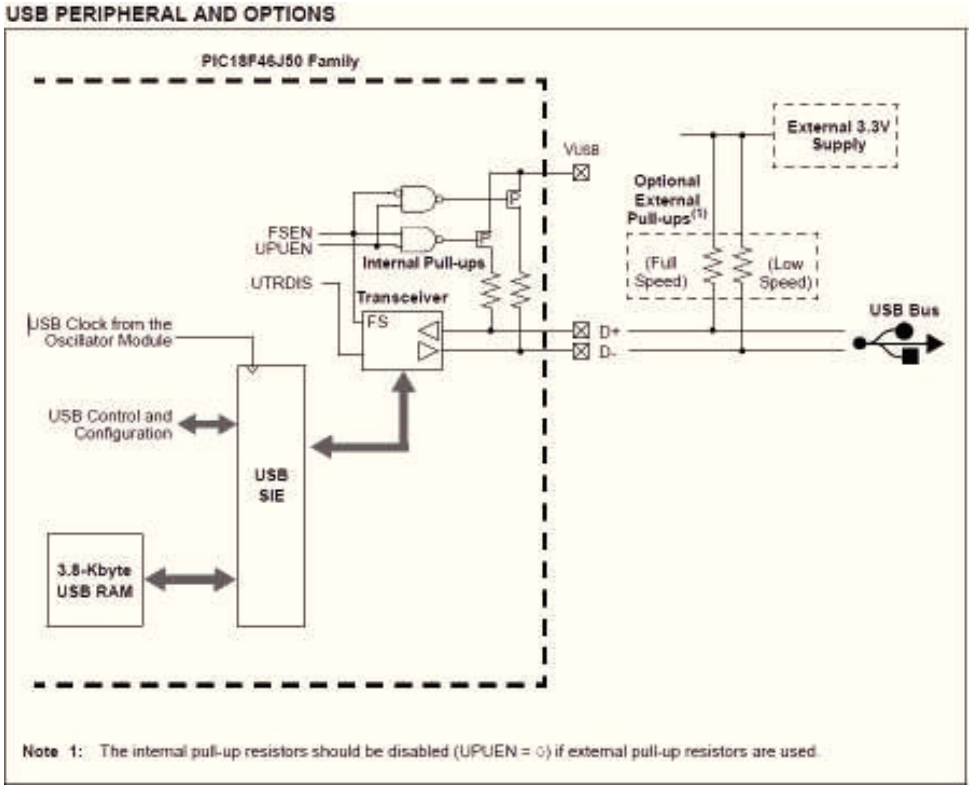
2.5.6.3 USB modul

Mikroprocesor PIC18F46J50 obsahuje USB Serial Interface Engine (SIE) kompatibilní modul podporující USB full-speed a low-speed přenosové rychlosti a umožňuje využít vysokorychlostní komunikaci mezi mikroprocesorem PIC a jakýmkoliv zařízením podporujícím funkci USB host . Díky vestavěnému internímu vysílači/přijímači může být USB SIE modul mikroprocesoru PIC přímo připojen ke sběrnici USB s minimem externích součástek. Pro zvýšení výkonu USB modulu byla integrována harwarová funkce duálního přístupu do datové paměti pro sdílení paměti mezi jádrem mikroprocesoru a mezi modulem SIE a dále byly integrovány deskriptory bufferů, které umožňují volně naprogramovat použití paměti pro USB endpointy. Funkce duálního přístupu do paměti byla nazvána (USB RAM).

Blokové schéma celé periférie mikroprocesoru PIC18F46J50 je na obrázku 6 „*USB periférie mikroprocesoru PIC18F46J50, zdroj: [6]*“ na straně 20.

Funkce modulu USB je ovládána pomocí tří řídících registrů a dalších devatenácti pomocných registrů ovládajících aktuální USB transakce. Tyto registry jsou:

- USB Control register (UCON)
- USB Configuration register (UCFG)
- USB Transfer Status register (USTAT)



Obrázek 6: USB periférie mikroprocesoru PIC18F46J50, zdroj: [6]

- USB Device Address register (UADDR)
- Frame Number registers (UFRMH:UFRML)
- Endpoint Enable registers 0 through 15 (UEPn)

2.5.6.4 USB RAM

Celý prostor 3.8K bajtů paměti RAM může být současně zpřístuněn jak jádrem mikroprocesoru tak i USB modulem SIE. Modul SIE používá oddělený USB DMA kontroler pro uložení všech příchozích packetů (OUT / SETUP) přímo do systémové datové paměti. Pro odchozí packety IN, modul SIE může přímo číst obsah SRAM a použít ho pro vytvoření data packetů, které posílá na USB host zařízení.

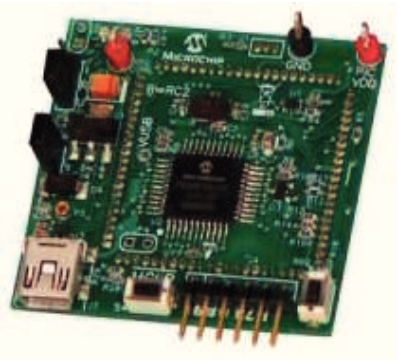
Ačkoliv je USB RAM přístupná mikroprocesoru jako datová paměť, k částem, které jsou využívány modulem USB SIE by nemělo být mikrokontrolérem přistupováno. Proto je implementován semaforový mechanismus, který řídí a vymezuje přístup k jednotlivým buferům. Deskriptory buferů BD potom definují velikost buferu endpointu a také jejich konfiguraci a řízení.

2.6 Popis nástrojů použitých při vývoji

2.6.1 Vývojový kit PIC18F46J50 FS USB Demo Board

Hotový vývojový kit jsem před vývojem na kontaktním poli upřednostnil z několika důvodů:

- Hotové zařízení připravené k okamžitému použití
- USB firmware bootloader, není nutný samostatný programátor
- Vyřešené napájení 3.3V, bezpečné připojení k programátoru a počítači
- Výrobcem otestované zařízení
- Celek, který lze snadno přepravit na jiné místo
- Přímá podpora v MCHPFSUSB frameworku výrobce



Obrázek 7: PIC18F46J50 FS USB Demo Board (MA180024), zdroj: [1]

PIC18F46J50 FS USB Demo Board (výrobní číslo MA180024) je USB2.0 full-speed a low-speed demonstrační a vývojová deska založená na mikroprocesoru PIC18F46J50. Výrobcem je doporučována jako výchozí USB vývojová platforma pro aplikace, které budou používat některý z rodiny mikroprocesorů PIC18F46J50 zahrnujíc typy PIC18F46J50, PIC18F45J50, PIC18F44J50, PIC18F26J50, PIC18F25J50 a PIC18F24J50.

Tato demo deska může pracovat buď zcela samostatně anebo může být pro rozšíření funkcionality vložena do populárního demo kitu PICDEM PIC18 Explorer Board (výrobní číslo DM183032). Pokud je demo deska používána samostatně, přiložený adaptér RJ-11 na ICSP lze použít k přeprogramování demo desky pomocí programátorů vybavených konektorem RJ-11. Deska je dodávána s nosičem CDROM obsahujícím příklady USB projektů, které lze přímo s deskou použít. Tyto příklady lze rovněž samostatně stáhnout ze

stránek výrobce, jsou součásti Microchip Aplication Library MCHPFSUSB frameworku. Demo deska je předprogramována demonstračním firware a také USB bootloaderem pro snadné přehrávání zkompilovaného firmware. USB bootloader je nezávislý na demo firmware a může být použit pro naprogramování nového firmware souboru *.hex do Flash paměti mikroprocesoru, čímž se eliminuje použití samostatného In-Circuit Serial Programming™ ICSP™ programátoru například PICkit 3.

Další informace lze získat na webových stránkách výrobce [1] a v uživatelském manuálu výrobce k demo desce [15]. Vyobrazení vývojové desky je na obrázku 7 „*PIC18F46J50 FS USB Demo Board (MA180024), zdroj: [1]*“ na straně 21.

2.6.2 Převodník úrovní PmodRS232

Deska PmodRS232 výroce Digilent je vybavena integrovaným obvodem ST3232, který zabezpečuje dvoucestný převod napěťových úrovní TTL modulu EUSART na napěťové úrovně sériového rozhraní RS-232. Tento obvod je v 3.3V verzi a je tedy přímo připojitelný k 3.3V verzi PIC18F46J50 FS USB Demo Board. Napěťové úrovně portu RS-232 jsou -3 až -12V pro logickou 1 a +3 až +12V pro logickou 0. PmodRS232 modul je konfigurován jako komunikační (DCE) zařízení a připojuje se k terminálnímu (DCE) zařízení, jako například k sériovému portu počítače, přes přímý kabel. Deska převodníku napěťových úrovní PmodRS232 je vyobrazena na obrázku 8 „*Deska převodníku napěťových úrovní PmodRS232, zdroj: [17]*“ na straně 22.

Obrázek 8: Deska převodníku napěťových úrovní PmodRS232, zdroj: [17]

Kompletní specifikaci desky najdete v manuálu [17] výrobce.

2.6.3 Programátor PICkit 3

Jako programátor používám PICkit 3 In-Circuit Debugger/Programmer Part Number: PG164130 od výrobce Microchip. Tento programátor se připojuje přímo do cílového obvodu laděného nebo programovaného mikroprocesoru a umí programovat všechny PIC mikroprocesory. K počítači se připojuje přes full-speed USB, umí si upgradovat firmware a má spoustu dalších užitečných funkcí, jako například ladění v reálném čase, zastavení periferií na breakpointu nebo užitečnou funkci programování bez počítače Programmer-to-Go. Další zajímavé vlastnosti lze nalézt v dokumentaci [14] k tomuto programátoru. Programátor je vyobrazen na obrázku 9 „*Programátor PICkit 3, zdroj: [1]*“ na straně 23.

Obrázek 9: Programátor PICkit 3, zdroj: [1]

I přes všechny zajímavé vlastnosti zmíněného programátoru jsem jej nakonec při vývoji emulátoru nepoužil, neboť vývojový kit PIC18F46J50 FS USB Demo Board používá svůj vlastní USB bootloader, který umí nahrát nový firmware do mikroprocesoru přímo z počítače přes USB rozhraní USB HID zařízení. Nicméně pokud bych při vývoji poškodil nebo přepsal bootloader uložený v mikroprocesoru vývojového kitu FS USB Demo Board, PICkit 3 by mi byl dobrým pomocníkem pro obnovení firmware vývojového kitu včetně zapsání původního USB bootloaderu do paměti flash mikroprocesoru.

2.6.4 Referenční FTDI převodník KU2-232

Abych mohl kromě teoretických informací získat i praktický přehled o chování převodníku USB-RS232 výrobce FTDI, který jsem měl za úkol emulovat, zahrnul jsem do vývoje také práci se skutečným hardwarovým převodníkem výrobce PremiumCord založeným na obvodu FTDI typu FT232R.

Poznámka 2.1 Originální převodník FTDI typu FT232R osobně vlastním a jsem s jeho chováním a funkcí dobře seznámem. Mám tedy základní přehled o tom, co očekávat

Obrázek 10: Originál USB-RS232 FTDI převodník, zdroj: [19]

od emulátoru tohoto převodníku. Převodník je vyobrazen na obrázku 10 „*Originál USB-RS232 FTDI převodník, zdroj: [19]*“ na straně 24 a jeho celý popis pro případ nákupu je v referencích [19].

2.6.5 Vývojové prostředí MPLAB® X IDE

Pro vývoj aplikací na mikroprocesorech Microchip se dlouhé roky používá integrované vývojové prostředí MPLAB IDE. V roce 2011 však Microchip uvolnil první funkční verzi 1.0 nového integrovaného vývojového prostředí MPLAB® X IDE, které od roku 2011 posbíralo již několik významných ocenění, například „Elektra Electronic Industry Award 2011“ nebo „ECN Readers' Choice Tech Award 2011“.

Vzhledem k tomu, že výrobce Microchip toto vývojové prostředí silně podporuje oproti starému vývojovému prostředí MPLAB v8, provedl jsem celý vývoj emulátoru v novém vývojovém prostředí MPLAB® X IDE verze 1.0. Uvedl bych zde některé vlastnosti, s ostatními se lze seznámit na webových stránkých výrobce Microchip. Vývojové prostředí MPLAB® X IDE je postaveno na platformě NetBeans, dědí tedy některé zajímavé vlastnosti jako open-source, plug-ins, cross-platform, několik verzí kompilátoru v jednom projektu, verzování, dokončování kódu, formátování kódu, barevné zvýrazňování kódu, vyhledávání funkcí, přejmenovávání atd.

2.6.6 Softwarové vývojové nástroje

2.6.6.1 Microchip C18 Compiler

Součástí zadání této práce je požadavek realizace emulátoru v jazyku C s použitím USB knihoven výrobce. Pro projekt jsem použil komiplátor jazyka C přímo od výrobce Microchip určený pro mikroprocesory řady PIC18. Název kompilátoru je Microchip C18 Compiler a byl použit ve verzi 3.40. Popis knihovny kompilátoru C18 lze najít v dokumentaci [7]. Další dokumentace je přítomna vždy v instalačním adresáři ke každé verzi komiplátoru.

2.6.6.2 Microchip USB framework

Pro USB implementaci jsou dostupné knihovny výrobce Microchip v balíku MCHPFSUSB Framework. Tento distribuovaný balík obsahuje USB projekty firmware pro řady mikroprocesorů PIC18, PIC24F, PIC32 včetně souvisejících USB ovladačů a zdrojů pro použití s počítačem. Podporuje USB na 8-bit, 16-bit a 32-bit mikroprocesorech ve full-speed (12 Mbps) rychlostech a podporuje USB režimy zařízení, host a On-The-Go. Tento software je poskytován zdarma včetně zdrojových kódů a obsahuje také příklady projektů, například Device CDC demo, Printer demo, bar code scanner demo, CDC serial emulator, device composite HID and mass storage, generic driver demo, HID mouse demo, HID keyboard demo, SD card reader, SD data logger a další. Specifikace a uživatelská příručka frameworku je k dispozici v referenci [9]. Kromě této příručky je veškerá aktuální dokumentace součástí instalace USB frameworku.

Pro projekt emulátoru byl použit USB framework MCHPFSUSB ve verzi v2.9c.

2.6.6.3 USB analyzátor USBTrace

Během vývoje emulátoru jsem postupně narazil na části, které nepracovaly jak jsem původně zamýšlel. Abych se ujistil, co se s vyvíjeným zařízením děje a jak navenek komunikuje po USB sběrnici, potřeboval jsem analyzátor sběrnicové komunikace. Protože jsem však nezamýšlel použít hardwarový analyzátor komunikace, použil jsem při vývoji softwarový analyzátor komunikace po sběrnici USB. Použil jsem volně dostupnou omezenou verzi programu USBTrace, jejímž poskytovatelem je společnost SysNucleus, viz. reference [18]. I přes svá omezení mi však program USBTrace svými funkcemi plně dostačoval k činnostem které jsem potřeboval. USBTrace program mi významně pomohl k odhalení specifické komunikace FTDI obvodu a porozumění teorie komunikace po USB sběrnici.

Program USBTrace je snadno ovladatelný a výkonný USB analyzátor. USBTrace může sledovat provoz na USB rozbočovači nebo hostitelském řadiči přímo na zařízení. USBTrace

je plně softwarový produkt, který podporuje operační systémy Windows 2000 až Windows 8 Beta a pracuje se secifikacemu USB 1.x, 2.0 a USB 3.0.

2.6.6.4 Hyperterminál

Pro konečné testování emulátoru byl použit terminální program „Hyperterminál“. Tento program není již součástí Windows 7, ale lze jej získat například z Windows XP jako spustitelný soubor. V ostatních operačních systémech byl použit obdobný program (terminál) specifický pro daný operační systém.

2.6.6.5 Ovladače USB

Pro vývoj bylo nutno nainstalovat ovladače Microchip pro spuštění demo aplikací z Microchip USB frameworku a ovladače pro FTDI USB převodník. Microchip ovladače jsou součástí Microchip USB frameworku v podobě „mchpcdc.inf“ souboru, jejímž nainstalováním se emuluje virtuální COM port, který má ovladače dostupné přímo v operačním systému. Pro FTDI USB převodník byly nainstalovány ovladače Virtual COM Port (VCP), které taktéž emulují standardní sériový COM port počítače. Tyto ovladače jsou volně dostupné na stránkách výrobce ke stažení, viz. refrerence [2].

2.6.6.6 Operační systém a platforma

Celý vývoj emulátoru jsem provedl pod licencovaným operačním systémem Windows 7 Professional 32bit na platformě Intel® Core™ i5 CPU, M520@2.40GHz se 4GB operační paměti RAM, notebook Dell Latitude®.

3 Převodník USB-RS232 firmy FTDI

3.1 Future Technology Devices International Ltd. ©

Společnost FTDI z Velké Británie (GB) je specialistou na převod periferních zařízení na sběrnici Universal Serial Bus (USB). Společnost nabízí řešení pro nejsnazší cestu k migraci na USB tím, že nabízí křemíkové součástky sériového USB RS232 a paralelního USB FIFO převodníku s podporou "ready-to-go"autorsky volných USB ovladačů. Celkové řešení založené na obvodech od společnosti FTDI pak nabízí snížené náklady na vývoj a ladění a které ve finále zapezpečují rychlé uvedení výrobku na trh.

3.2 Typy převodníků

Výrobce FTDI nabízí velké množství obvodů pro převod periférií na sběrnici USB. Jejich celkový přehled naleznete na webových stránkách výrobce FTDI, viz. zdroj [13].

Pro převod sběrnice USB na sběrnici RS-232 (UART) nabízí výrobce FTDI tyto převodníky v kategoriích vysokorychlostní, standardní a výrobně ukončené:

1. High speed
   - FT232H - Single Channel Hi-Speed USB to Multipurpose UART/FIFO IC
   - FT2232H - Hi-speed USB 2.0 Slave to Dual UART/FIFO Converter
   - FT4232H - Hi-speed USB 2.0 Slave to Quad Channel UART / Serial Converter
2. Standart
   - FT2232D - USB 2.0 Slave to Dual UART / FIFO Converter
   - FT232R - USB 2.0 Slave to UART Converter
   - FT232B - USB 2.0 Slave to UART Converter (FT232R version recommended for new designs)
3. Discontinued
   - FT2232L - Dual USB UART/FIFO IC (lead free version of FT2232C) - please see FT2232D for a drop-in replacement.
   - FT2232C - Dual USB UART/FIFO IC - please see FT2232D for a drop-in replacement.
   - FT232BM - USB UART IC - please see FT232BL for a drop-in replacement.
   - FT8U232AM - USB UART IC - not recommended for new designs - use the FT232R instead.

Doporučením vedoucího práce bylo vybrat k emulaci některý z novějších typů převodníku FTDI, aby byla zajištěna jeho budoucí podpora jak na trhu tak v ovladačích výrobce. Protože High speed FTDI převodníky jsou postaveny na rychlosti (USB) 480Mbps a (UART) do 12Mbaud, přičemž ani jedné rychlosti nelze s vybraným mikroprocesorem PIC18F46J50 dosáhnout, tato skupina vypadla z výběru. Stejně, tedy jejím vyřazením, dopadla i skupina výrobcem FTDI již nevyráběných typů. V poslední skupině jsem omezil výběr na jednoportový UART převodník a protože verze „B“ není doporučena pro nové konstrukce a s přihlédnutím k FTDI převodníku, který již vlastním a je založen na stejném FTDI chipu, jsem zvolil jako vhodný převodník pro emulaci typ FT232R.

Přehled vlastností jednoportových UART převodníků výrobce FTDI je uveden na obrázku 11 „*UART obvody FTDI, zdroj: [2]*“ na straně 29.

FT Device	FT232H	FT232R	FT232B
			
Description	Single Channel Hi-Speed USB to Multipurpose UART/FIFO IC	USB 2.0 Slave to UART Converter	USB 2.0 Slave to UART Converter (FT232R version recommended for new designs)
USB Speed	High (480Mbps) / Full Speed (12Mbps)	Full Speed (12Mbps)	Full Speed (12Mbps)
No. USB Ports	1	1	1
No. External Channels	1	1	1
Supported External Interfaces	UART, FIFO, 1 x MPSSE*, Fast serial, 8051 MCU emulation, FT1248	UART with 4 GPIO pins	UART
Internal Memory	1kB RX/TX buffer per channel	128 Byte – RX 256 Byte – TX	348 Byte – RX 128 Byte – TX
UART Speeds	Up to 12Mbaud	Up to 3Mbaud	Up to 1Mbaud
Configuration Data Storage	Via External EEPROM	Internal EEPROM	Via External EEPROM
Operating Temperature	-40°C to +85°C	-40°C to +85°C	0°C to +70°C
Packages	48-pin LQFP 48-pin QFN	32-pin QFN 28-pin SSOP	32-pin LQFP

*MPSSE - Multi-Protocol Synchronous Serial Engine, configurable serial controller designed to support SPI, I2C, JTAG, and other serial interfaces at speeds of up to 30Mbps.

Obrázek 11: UART obvody FTDI, zdroj: [2]

3.3 Převodník typ FT232R technický popis

Poznámka 3.1 Vlastnosti obvodu FT232R, obrázky, schemata a další informace uvedené v této kapitole pocházejí z informací veřejně publikovaných na webových stránkách výrobce FTDI [2], z internetu a ze zdroje [20]. Uváděny jsou pouze vlastnosti potřebné k emulaci převodníku z USB na sériový UART RS-232 port. Kompletní vlastnosti obvodu FT232R může zájemce nalézt v originální dokumentaci výrobce, viz. zdroj [3].

Podle sdělení výrobce (k březnu 2012) na jeho webových stránkách je převodník FT232R „the latest device“ - jeho nejposlednější zařízení přidané do skupiny USB UART integrovaných obvodů.

FTDI obvod FT232R je převodník z USB na sériový UART s volitelným výstupem generátoru a s novou vlastností FTDIChip-ID^{TM}. Doplňkově byly do obvodu přidány funkce asynchronního a synchronního bit-bang rozhraní.

Výrobce FTDI přidal do svého obvodu v porovnání se svými předchůdci dvě nové funkce a vytvořil tak zařízení typu „3 v 1“. Jednou z funkcí je přesný interní generátor hodinového kmitočtu (6MHz, 12MHz, 24MHz a 48MHz), který může být vyveden vně obvodu FT232R a použit pro řízení externího mikroprocesoru nebo vnějšího logického obvodu. Druhou nově integrovanou funkcí je jedinečné číslo (the FTDIChip-ID^{TM}), které je vypáleno do zařízení při výrobním procesu po otestování samotného obvodu. Toto jedinečné číslo je čitelné přes sběrnici USB, čímž je možno obvod FT232R použít jako základ bezpečnostní klíčenky, která může být následně použita například k ochraně zakázkových programů proti kopírování, tzv. hardwarový klíč.

Konstrukce používající obvod FT232R se díky integrovené paměti EEPROM, integrovaným USB rezistorům a integrovanému obvodu hodinového signálu, který nepotřebuje žádný externí krystal, dále zjednodušují.

3.3.1 Vybrané vlastnosti

Zde uvedené vlastnosti se týkají části UART převodníku FT232R.

- Jednoobvodové řešení, integrovaný přesný interní oscilátor, integrovaná 1024bit paměť EEPROM, integrované USB rezistory
- EEPROM obsahuje USB VID, PID, USB sériové číslo a řetězec popisu produktu
- USB sériové číslo je jedinečné, předprogramované výrobcem
- UART podporuje 7 nebo 8 data bitů, 1 nebo 2 stop bity a paritu (odd / even / mark / space / no parity)

- Hardwarový (RTS#/CTS#) nebo softwarový XON, XOFF handshaking
- Data transfer 300 baud až 1 Megabaud (RS232)
- Adjustable receive buffer timeout (Latency Timer)
- USB 2.0 Full Speed kompatibilní
- FIFO receive and transmit buffers for high data throughput
- 256 Byte receive buffer and 128 Byte transmit buffer utilising buffer smoothing technology to allow for high data throughput
- USB bulk transfer mode
- In-built support for event characters and line break condition
- Napájecí napětí 3.3V až 5.25V

3.3.2 Blokové schéma

Jednotlivé části obvodu FT232R jsou zobrazeny v blokovém schématu na obrázku 12 „*Blokové schéma obvodu FT232R, zdroj: [20]*“ na straně 32. Níže si popíšeme vybrané bloky.

Vnitřní paměť EEPROM. Ve vnitřní paměti jsou uchovány USB informace jako Vendor ID (VID), Product ID (PID), device serial number, řetězec popisu produktu, uživatelské informace a konfigurace obvodu FT232R.

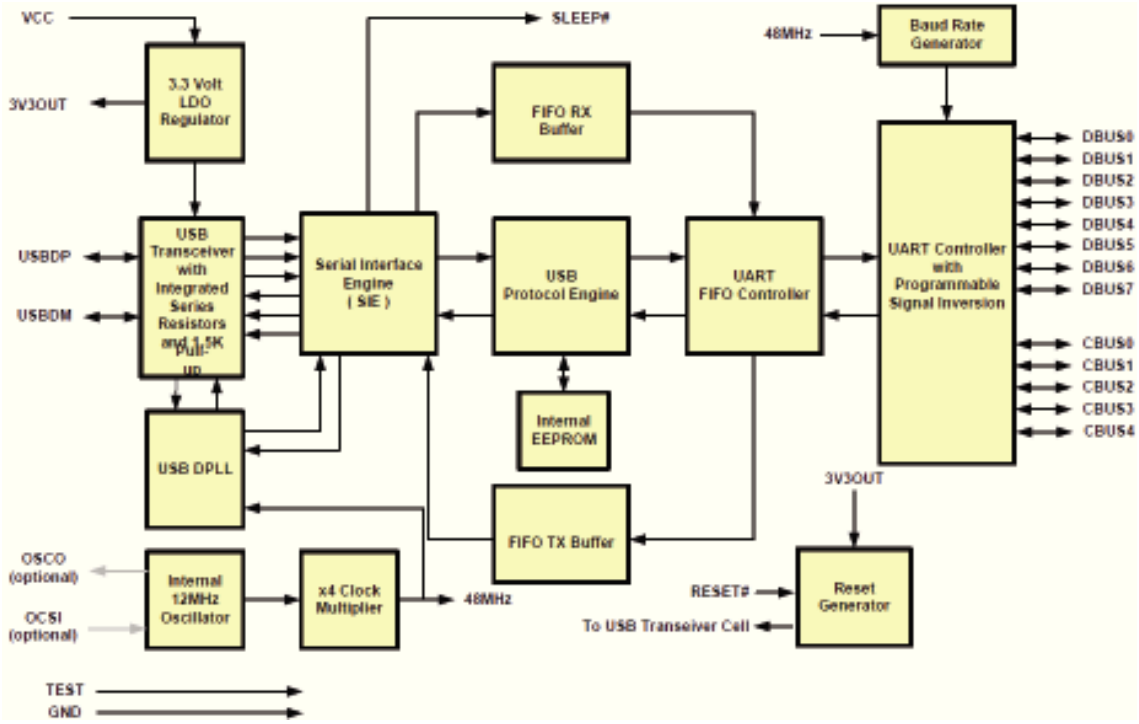
Serial Interface Engine (SIE). SIE provádí konverzi sériových dat na paralelní a paralelních dat na sériová. Dále dle specifikace USB 2.0 provádí bit stuffing/un-stuffing a generování CRC5/CRC16.

USB Protocol Engine. USB Protocol Engine řídí datové proudy z USB řídícího endpointu. Zpracovává nízkoúrovňové požadavky USB host kontroleru a požadavky kapitoly 9 specifikace USB 2.0.

FIFO RX Buffer (128 bytes). Data poslaná z USB host kontroleru přes USB data OUT endpoint jsou ukládána v FIFO RX buferu. Data se z buferu odebírají na základě řízení UART FIFO kontroleru.

FIFO TX Buffer (256 bytes). Data z UART přijímacího registru jsou ukládána do TX buferu. USB host kontroler odebírá data z FIFO TX buferu zasláním USB požadavku z IN endpointu.

UART Controller with Programmable Signal Inversion and High Drive. UART kontroler dohromady s UART FIFO kontrolerem řídí přesun dat mezi FIFO RX a FIFO



Obrázek 12: Blokové schéma obvodu FT232R, zdroj: [20]

TX bufery a UART Tx a UART Rx registry. Provádí sérioparalelní konverzi dat a řídí handshaking a signály portu RS232.

Obvod FT232R byl navržen pro efektivní spolupráci s USB host kontrolerem tak, aby používal co nejmenší možnou šířku USB pásma.

Windows OS	Other OS
Windows Server 2008 R2	Linux
Windows 7 (x64)	Mac OS X
Windows Server 2008 (x64)	Mac OS 9
Windows Server 2003 (x64)	Mac OS 8
Windows Vista (x64)	Android
Windows CE.NET (od v4.2)	
Windows XP (x64)	
Windows ME	
Windows 98	

Tabulka 1: Podporované operační systémy FTDI, zdroj: [2]

3.3.3 Ovladače FTDI

Výrobce FTDI nabízí dva druhy ovladačů svých USB převodníků.

Virtual COM Port (VCP) ovladače. VCP ovladače emulují standardní sériový COM port počítače. Ve správcí zařízení se vytvoří nový COM port se kterým se potom komunikuje jako by to bylo standardní RS232 zařízení.

D2XX ovladače. Ovladače D2XX umožňují přímý přístup na USB zařízení přes rozhraní dll souboru pomocí volání jeho funkcí. Popis ovládání USB zařízení přes ovladač D2XX je výrobcem zdokumentován a poskytován zdarma včetně příkladů použití.

Podporované operační systémy pro USB převodníky společnosti FTDI jsou shrnuty v tabulce 1 *„Podporované operační systémy FTDI, zdroj: [2]“* na straně 33.

3.4 Převodník typ FT232R programátorský popis

3.4.1 Dostupnost informací

Bohužel se mi z dostupných zdrojů internetu nepodařilo získat podrobné informace o vnitřní práci obvodu FT232R, zejména pak specifikaci protokolu, s jakým toto zařízení komunikuje se svými FTDI ovladači.

Nepodařilo se mi z dostupných zdrojů internetu získat také žádné podrobné informace o podobném projektu, který by v mikroprocesoru emulovat převodník FTDI. Pokud je mi tedy známo, je toto zatím první projekt emulátoru svého druhu.

Na základě informačního embarga jsem se tedy pokusil o nahlédnutí do tajů operačního systému Linux, který mi není zcela vlastní a jako prvotní zdroj informací jsem použil ovladače FTDI napsané pod operačním systémem Linux, jejichž zdrojové kódy jsou volně dostupné.

Nakonec tedy tato kapitola vznikla za účasti pramenů ovladače FTDI ze systému Linux, z nesčetných pokusů, omylů a zkušeností při psaní firmware emulátoru a také za pomoci analyzátoru USBTrace, který mi umožnil pochopit některé speciality, které jsem na první pohled nebyl schopen z ovladače operačního systému Linux pochopit, a jenž mi umožnil udělat téměř identickou kopii protokolu, jako má referenční FTDI převodník KU2-232 viz. zdroj [19], porovnáním chování vyvíjeného firmware emulátoru v mikroprocesoru PIC18F46J50 a zmíněného referenčního převodníku.

Pro úplnost úvodu k této kapitole zde uvádím zdroje, které jsem použil jako základní informační kapitál pro napsání firmware emulátoru obvodu FTDI FT232R v mikroprocesoru Microchip PIC18F46J50. Zdroje jsou [21, 22, 23, 24, 25] a [26].

Dále již kapitola obsahuje konkrétní informace ze specifikace USB a vyžaduje přiměřené znalosti z této oblasti. Těmito informacemi se nebudeme v této práci detailněji zabývat, případně je okrajově zmíníme bez náležitých detailů. Pro úplnost však je možné je dohledat v referenci [12] nebo [27].

Poznámka 3.2 Některé informace uváděné v této kapitole by si jistě zasloužily spíše místo v kapitole 3.3 „*Převodník typ FT232R technický popis*“ na straně 30. Jsou však ale záměrně umístěny právě zde, protože pochází buď z výše uvedených zdrojů nebo jsou zde uvedeny na základě vlastních experimentů při vývoji emulátoru a nemohou tedy patřit do všeobecně známých nebo výrobcem uváděných vlastností obvodu FT232R. Do této skupiny informací patří zejména vnitřní pamět EEPROM a Latency Timer.

3.4.2 Enumerace

Poznámka 3.3 Níže za „ovladač“ budeme označovat ovladač FTDI pro zařízení FT232R, za „zařízení“ označovat samotný obvod FT232R a jeho emulovanou verzi v mikroprocesoru PIC18F46J50 budeme označovat „emulované zařízení“.

Enumeraci definuje specifikace USB jako automatické rozpoznání připojeného zařízení a zavedení ovladačů odpovídajících tomuto zařízení do paměti operačního systému.

Rozpoznání obvodu FT232R, který se připojí ke sběrnici USB počítače prochází třemi fázemi na úrovni komunikace obvodu FT232R, ovladače FTDI a samozřejmě operačního systému.

Všechny tři výše uvedené fáze se ověřují a při selhání kterékoliv z nich není zařízení rozpoznáno a není vytvořen virtuální port ve *Správci zařízení*. Fáze jsou následující:

1. Oveření deskriptoru operačním systémem a nahrání správného ovladače
2. Ovladač dotazuje informace z vnitřní paměti EEPROM obvodu FT232R
3. Ovladač dotazuje/nastavuje základní informace/parametry obvodu FT232R
4. Zařízení je rozpoznáno a začíná jeho práce

Čtvrtá fáze je zde uvedena pro úplnost, aby bylo zcela jasné, kdy začíná samotná práce zařízení, ke které bylo svojí funkcí určeno.

Detailní proces enumerace podle rozčlenění do jednotlivých fází pak vypadá následovně:

1. Operační systém (OS)
   - OS si vyžádá prvních 18 bajtů deskriptoru (DEVICE)
   - OS si vyžádá zbytek deskriptoru (CONFIGURATION)
   - OS vybírá konfiguraci zařízení
2. Ovladač (OVL) - čtení EEPROM
   - OVL se dotazuje na obsah EEPROM (Request 0x90h), zařízení odpovídá postupně po 2 bajtech obsahem paměti
3. Ovladač (OVL) - dotaz/nastavení
   - OVL si vyžádá část deskriptoru (STRING), konkrétně sériové číslo zařízení

- OVL nastaví vlastnosti linky RS232 (Request 0x04h)
- OVL nastaví DTR_HIGH (Request 0x01h)
- OVL nastaví RTS_HIGH (Request 0x01h)
- OVL nastaví flow control linky RS232 (Request 0x02h)
- OVL nastaví baudrate linky RS232 (Request 0x03h)
- OVL nastaví RTS_LOW (Request 0x01h)
- OVL nastaví DTR_LOW (Request 0x01h)
- OVL pošle požadavek na reset zařízení (6x Request 0x00h)
- OVL pošle požadavek na nastavení LatencyTimeru (Request 0x09h)

4. Zařízení (ZA)

   - ZA posílá automaticky v intervalu nastaveného LatencyTimeru status RS232 linek (BULK_TRANSFER)
   - OVL dále průběžně posílá požadavky dle své potřeby, na které ZA odpovídá

Po provedení posledního kroku ve třetí fázi je zařízení úspěšně operačním systémem rozpoznáno. Operační systém se dotazuje přes (CONTROL_TRANSFER) a zařízení odpovídá jako (VENDOR_DEVICE).

3.4.3 Deskriptor

Každé USB zařízení má svůj vlastní deskriptor (česky popis zařízení), kterým popisuje své vlastní schopnosti, jako například jakým typem zařízení je, kdo ho vyrobil, verzi USB, kterou podporuje, kolik má konfigurací, počtu endopintů a jejich typů.

Podle deskriptoru konkrétně jeho části Product ID (PID) a Vendor ID (VID) vybírá operační systém správný ovladač pro použití se zařízením a jeho konfigurací. V jednu chvíli může být aktivní jen jedna konfigurace zařízení.

V tabulce 2 „*Deskriptor FT232R (DEVICE), zdroj: [19]*“ na straně 37 je uveden deskriptor „zařízení “ referenčního zařízení USB převodníku KU2-232 viz. zdroj [19]. Stejný deskriptor byl použit v emulovaném zařízení.

Pole	Hodnota
VID	0x0403h
PID	0x6001h
Výrobce	FTDI
Produkt	USB Serial Converter
Sériové číslo	FTR69S36

Tabulka 2: Deskriptor FT232R (DEVICE), zdroj: [19]

V tabulce 3 „*Deskriptor FT232R (CONFIGURATION), zdroj: [19]*“ na straně 37 je uveden deskriptor „konfigurace zařízení“ referenčního zařízení USB převodníku KU2-232 viz. zdroj [19]. Stejný deskriptor byl použit v emulovaném zařízení.

Pole	Hodnota
Device Type	Vendor Specific
USB Class	0xFFh
USB SubClass	0xFFh
USB Protocol	0xFFh
USB Version	0x200h

Tabulka 3: Deskriptor FT232R (CONFIGURATION), zdroj: [19]

Poznámka 3.4 Výrobce programuje do interní EEPROM paměti obvodu FT232R jedinečné sériové číslo. Pokud takové zařízení odpojíte z USB portu a připojíte ho k jinému USB portu, operační systém nerozpozná zařízení jako nové, ale naopak mu přidělí číslo virtuálního COM portu jako mělo v USB portu, ze kterého bylo vytaženo. Toto chování lze v operačním systému Windows 7 nastavit jinak.

3.4.4 Endpointy

Endpointy (česky koncové body) jsou místa, do kterých míří data nebo ze kterých se data budou číst. Endpointy jsou fyzicky umístěny v připojeném zařízení, mohou být jednosměrné IN nebo OUT anebo obousměrné IN/OUT a mají svojí velikost v počtech bajtů, které mohou uchovávat. Pojmenování IN/OUT je z pohledu USB host zařízení (počítače). Endpointy se vyznačují svojí jednoznačnou adresou. Každé zařízení má minimálně jeden endpoint nula EP0, jehož prostřednictvím probíhají standardní požadavky (Requesty) s přenosem dat (CONTROL_TRANSFER).

Endpointy lze také vidět jako rozhraní mezi hardware zařízení a firmware na tomto zařízení běžící, které potom využívá USB host kontroler ke komunikaci s tímto připojeným zařízením.

Endpoint	Adresa	Směr	Přenos	Velikost [Byte]
EP0	0x00h	IN/OUT	(CONTROL_TRANSFER)	8
EP1	0x81h	IN	(BULK_TRANSFER)	64
EP2	0x02h	OUT	(BULK_TRANSFER)	64

Tabulka 4: Deskriptor FT232R (ENDPOINTY), zdroj: [19]

Endpointy obvodu FT232R jsou shrnuty v tabulce 4 „*Deskriptor FT232R (ENDPOINTY), zdroj: [19]*“ na straně 38. Přenos dat protokolem BULK_TRANSFER se vyznačuje možností přenášet velké množství dat s kontrolou jejich neporušenosti ale bez garantované šířky pásma a bez garantovaného zpoždění.

Z uvedené tabulky 4 „*Deskriptor FT232R (ENDPOINTY), zdroj: [19]*“ na straně 38 je také patrno, že obvod FTR232R používá pro každý směr přenosu dat jeden endpoint. Ze směru dat od USB hosta do zařízení používá obvod FT232R endpoint EP2 a pro data pocházející z opačného směru od zařízení do USB hosta používá endpoint EP1. Asi nebude na škodu když prozradím, že EP1 a EP2 se využívají výhradně pro přenos RS232 dat, s malou výjimkou, že po endpointu EP1 ze směru od zařízení jdou samozřejmě také data, která určují aktuální stav převodníku RS232 a stav portů RS232. Veškeré ostatní řídící signály, pokyny a requesty pro zařízení jsou směrovány po řídícím kanálu endpointu EP0.

Poznámka 3.5 Při testování firmware z pohledu endpointů jsem zjistil a ověřil, že endpointy nelze vybírat náhodně. Všechny použité endpointy v zařízení musí začínat endpointem EP1 a pokračovat výše podle jejich potřebného počtu. Nelze začít ani od EP2 a nelze například ani jeden v řadě vynechat. Je například nepřípustné definovat endpointy EP1, EP3, EP4. Takto definované zařízení a připojené k USB portu v horším případě nebude operačním systémem rozpoznáno nebo v lepším případě bude nefunkční.

3.4.5 Vnitřní paměť

Obvod převodníku FT232R má v sobě integrovanou paměť EEPROM. Ta slouží zejména pro uchování informací o zařízení, obsahuje tedy kompletní deskriptor zařízení. Paměť obsahuje CRC checksum.

Protože jsem nebyl schopen nikde dohledat konkrétní obsazení paměti, tedy co za informace a na jakých adresách jsou v EEPROM umístěny, zejména pak informaci o tom, ze kterých částí EEPROM se počítá „checksum“ a kde je umístěn, pro emulátor jsem použil celou kopii EEPROM referenčního zařízení KU2-232 [19].

Poznámka 3.6 Při pokusu změnit v EEPROM emulátoru sériové číslo bez přepočítaného součtu paměti „checksum“, ovladač toto odhalil a začal se opakovaně ptát znovu na GET_DESCRIPTOR a opakovaně si žádal znovu sériové číslo z EEPROM. Při použití originálního obsahu EEPROM převodníku KU2-232 [19] problémy s opakovaným requestem GET_DESCRIPTOR zmizely.

Poznámka 3.7 Pro vážné zájemce, výrobce FTDI v jednom ze svých datasheetů „Application Note AN_105 FTDI Device EEPROM Programming Using a Vinculum VNC1L“ uvádí „The EEPROM structure document is only available on request under NDA. The details of how to calculate the checksum of the EEPROM are in EEPROM documents that are only available by request under NDA (Non-disclosure Agreement).“

3.4.6 Latency Timer

LatencyTimer je časovač implementovaný do obvodu FT232R, který opakovaně vždy po svém vypršení od nastaveného času pošle USB host zařízení informaci o stavu převodníku RS232, stavu portů RS232 a pošle také všechna data dostupná v buferu portu RS232 pokud nějaká příchozí k odeslání jsou. Touto technologií se vyrovnávají buferové rozdíly v komunikaci USB versus RS232 a snižuje se použitá šířka pásma USB na nejnižší možnou míru.

Výchozí velikost časovače je 16ms a ovladač ji může dynamicky přizpůsobit množství přenášených dat a rychlosti přenosu zasláním requestu převodníku FT232R. Nastavitelná velikost je potom 1 až 255ms. Podrobná dokumentace k LatencyTimeru, Handshakingu a Buferingu je uvedena ve zdroji [4].

3.4.7 Popis USB protokolu FTDI - Control Requests

V této části bude popsán minimální soubor řídících požadavků ovladače (USB Control Requestů) na které musí emulátor odpovídat, aby mohl emulovat FTDI převodník FT232R. Všechny tyto requesty byly emulátorem implementovány.

V tabulkách níže jsou uvedeny názvy parametrů USB požadavků zařízení (bRequest, wValue atd.) jak jsou uvedeny v USB specifikaci [12].

3.4.7.1 SIO_RESET_REQUEST 0x00h

Požadavek 0x00h slouží k resetování obvodu FT232R. Skladba USB požadavku je uvedena v tabulce 5 „*Control Request 0x00h, zdroj: autor*“ na straně 41. Hodnota požadavku *wValue* upřesňuje typ resetování obvodu FT232R, jednotlivé účinky jsou pak následující:

1. wValue = 0
    - Sets flow control set to „none“
    - Event char = $0D
    - Event trigger = disabled
    - Purge RX buffer
    - Purge TX buffer
    - Clear DTR
    - Clear RTS
2. wValue = 1
    - Purge RX buffer
3. wValue = 2
    - Purge TX buffer

Požadavek nevrací žádná *Data* proto je délka požadavku *wLength* nulová. Hodnota *wIndex* určuje RS232 port, kterého se tento požadavek týká, má smysl pro víceportové obvody, například obvod FT2232D.

Rychlost portu a jeho nastavení se resetem obvodu FT232R nemění.

Poznámka 3.8 Nepodařilo se mi zjistit, zda-li požadavek „Purge RX/TX Buffer“ na vyčištění buferu znamená jeho smazání nebo jeho vyčištění okamžitým odesláním jeho obsahu. Vzhledem k tomu, že požadavek 0x00h obvodu FT232R přichází na konci jeho enumerace v operačním systému, vyložil jsem si pojem „Purge“ jako prosté smazání „vynulování “ RX a TX buferů a tento požadavek jsem takto implementoval.

Parametr	Hodnota
bRequest:	0x00h
wValue:	0 = Reset SIO, 1 = Purge RX buffer, 2 = Purge TX buffer
wIndex:	Port
wLength:	0
Data:	none

Tabulka 5: Control Request 0x00h, zdroj: autor

3.4.7.2 SIO_SET_MODEM_CTRL_REQUEST 0x01h

Požadavek 0x01h slouží k nastavení linek RTS a DTR portu RS232. Skladba USB požadavku je uvedena v tabulce 6 „*Control Request 0x01h, zdroj: autor*“ na straně 41. Hodnota požadavku *wValue* upřesňuje typ operace nad linkou RTS nebo DTR, jednotlivé účinky jsou pak následující:

1. wValue = 0x101
    - Set DTR high
2. wValue = 0x100
    - Set DTR low
3. wValue = 0x201
    - Set RTS high
4. wValue = 0x200
    - Set RTS low

Požadavek nevrací žádná *Data* proto je délka požadavku *wLength* nulová. Hodnota *wIndex* určuje RS232 port, kterého se tento požadavek týká, má smysl pro víceportové obvody, například obvod FT2232D.

Parametr	Hodnota
bRequest:	0x01h
wValue:	0x101 DTR_HIGH, 0x100 DTR_LOW, 0x202 RTS_HIGH, 0x200 RTS_LOW
wIndex:	Port
wLength:	0
Data:	none

Tabulka 6: Control Request 0x01h, zdroj: autor

3.4.7.3 SIO_SET_FLOW_CTRL_REQUEST 0x02h

Požadavek 0x02h slouží k nastavení způsobu řízení dat linky RS232 tzv. „handshakingu“. Skladba USB požadavku je uvedena v tabulce 7 „*Control Request 0x02h, zdroj: autor*“ na straně 42.

Hodnota požadavku *wIndex* upřesňuje typ handshakingu, jednotlivé účinky jsou pak následující:

1. wIndex = 0x000
   - No handshaking
2. wIndex = 0x0100
   - RTS_CTS handshaking
3. wIndex = 0x0200
   - DTR_DSR handshaking
4. wIndex = 0x0400
   - XON_XOFF handshaking

Hodnota požadavku *wValue* definuje znaky v případě použití XON/XOF handshakingu následovně:

1. hValue
   - Znak XOFF
2. lValue
   - Znak XON

Parametr	Hodnota
bRequest:	0x02h
wValue:	Znak XOFF/XON
wIndex:	hIndex = Protocol, lIndex = Port
wLength:	0
Data:	none

Tabulka 7: Control Request 0x02h, zdroj: autor

Požadavek nevrací žádná *Data* proto je délka požadavku *wLength* nulová. Hodnota *lIndex* určuje RS232 port, kterého se tento požadavek týká, má smysl pro víceportové obvody, například obvod FT2232D.

Poznámka 3.9 Prefixy [h, l] před parametry požadavků značí horní a dolní část bajtu parametru. Například *hIndex* značí horní bajt parametru *wIndex* a *lIndex* značí dolní bajt parametru *wIndex*. Prefix [w] před parametrem samozřejmě značí velikost parametru v bajtech, například *wIndex* značí, že parametr „Index“ má velikost dva bajty.

3.4.7.4 SIO_SET_BAUDRATE_REQUEST 0x03h

Požadavek 0x03h slouží k nastavení rychlosti dat linky RS232 tzv. „baudrate“. Skladba USB požadavku je uvedena v tabulce 9 „*Control Request 0x03h, zdroj: autor*“ na straně 43. Hodnota požadavku *wValue* určuje požadovanou komunikační rychlost baudrate, jednotlivé hodnoty *wValue* a příslušné baudrate jsou uvedeny v tabulce 8 „*FTDI BaudRates, zdroj: autor*“ na straně 43.

wValue	Baud Rate
0x2710h	300
0x1388h	600
0x09C4h	1200
0x04E2h	2400
0x0271h	4800
0x4138h	9600
0x809Ch	19230
0xC04Eh	38461
0x0034h	57692
0x001Ah	115384
0x000Dh	230769
0x4006h	461538
0x8003h	923076
0x0000h	RESERVED
0x80D0h	14406

Tabulka 8: FTDI BaudRates, zdroj: autor

Poznámka 3.10 Zde jsou uváděny standardní hodnoty baudrate. Čip FT232R však podporuje i nestandardní hodnoty baudrate, podrobnosti lze získat v literatuře [5].

Parametr	Hodnota
bRequest:	0x03h
wValue:	BaudRate value
wIndex:	Port
wLength:	0
Data:	none

Tabulka 9: Control Request 0x03h, zdroj: autor

Požadavek nevrací žádná *Data* proto je délka požadavku *wLength* nulová. Hodnota *wIndex* určuje RS232 port, kterého se tento požadavek týká, má smysl pro víceportové obvody, například obvod FT2232D.

Poznámka 3.11 Emulátor zpracovává nastavení rychlosti maximálně do 115kbaud, protože vyšší rychlosti není schopen hardwarově zpracovat.

3.4.7.5 SIO_SET_DATA_REQUEST 0x04h

Požadavek 0x04h slouží k nastavení kódování dat komunikační linky RS232 tzv. „Parita, Data bity, Stop bity“. Skladba USB požadavku je uvedena v tabulce 10 „*Control Request 0x04h, zdroj: autor*“ na straně 44. Hodnota požadavku *wValue* je následující:

1. wValue (Bits 0-7)
   - Number of data bits
2. wValue (Bits 8-10)
   - Parity: NONE=0x00, ODD=0x01, EVEN=0x02, MARK=0x03, SPACE=0x04
3. wValue (Bits 11-12)
   - Stop bits: STOP_BIT_1=0x00, STOP_BIT_15=0x01, STOP_BIT_2=0x02

Parametr	Hodnota
bRequest:	0x04h
wValue:	Line Coding
wIndex:	Port
wLength:	0
Data:	none

Tabulka 10: Control Request 0x04h, zdroj: autor

Požadavek nevrací žádná *Data* proto je délka požadavku *wLength* nulová. Hodnota *wIndex* určuje RS232 port, kterého se tento požadavek týká, má smysl pro víceportové obvody, například obvod FT2232D.

3.4.7.6 SIO_POLL_MODEM_STATUS_REQUEST 0x05h

Požadavkem 0x05h si ovladač nárokuje navrácení stavu řídících linek portu RS232 tzv. „DSR / CTS / RI / RLSD(DCD)“ a stavu modemu „vnitřní části SIO obvodu FT232R“.

Požadavek se vrací jako dvoubajtová hodnota zařízení, kde první bajt je status linek a druhý bajt je status modemu. Obvod FT232R vrací tento dvoubajtový požadavek také jako první dva bajty přijatých dat portu RS232, at'už na základě požadavku ovladače nebo automaticky po vypršení Latency Timeru. Skladba USB požadavku je uvedena v tabulce 11 „*Control Request 0x05h, zdroj: autor*“ na straně 45. Hodnota navráceného požadavku parametru *Data* je následující:

1. Data[0] - bajt 0 line status
    - (Bit 0) Reserved - must be 1
    - (Bit 1-3) Reserved - must be 0
    - (Bit 4) (CTS) Clear to Send, 0 = inactive, 1 = active
    - (Bit 5) (DSR) Data Set Ready, 0 = inactive, 1 = active
    - (Bit 6) (RI) Ring Indicator, 0 = inactive, 1 = active
    - (Bit 7) (RLSD) Receive Line Signal Detect = „Data Carrier Detect (DCD)“, 0 = inactive, 1 = active
2. Data[1] - bajt 1 modem status
    - (DR) Data Ready
    - (OE) Overrun Error
    - (PE) Parity Error
    - (FE) Framing Error
    - (BI) Break Interrupt
    - (THRE) Transmitter Holding Register
    - (TEMT) Transmitter Empty
    - Error in RCVR FIFO

Požadavek vrací *Data* odpovídající stavu řídících linek a stavu modemu, délka požadavku *wLength* jsou dva bajty (slovo). Hodnota *wIndex* určuje RS232 port, kterého se tento požadavek týká, má smysl pro víceportové obvody, například obvod FT2232D.

Parametr	Hodnota
bRequest:	0x05h
wValue:	0
wIndex:	Port
wLength:	1
Data:	Line & Modem Status

Tabulka 11: Control Request 0x05h, zdroj: autor

3.4.7.7 SIO_SET_LATENCY_TIMER_REQUEST 0x09h

Požadavek 0x09h slouží k nastavení hodnoty vnitřního Latency Timeru obvodu FT232R. Hodnota je v jednotkách milisekund [ms] a je nastavitelná v rozsahu 1 až 255ms. Skladba USB požadavku je uvedena v tabulce 12 „*Control Request 0x09h, zdroj: autor*“ na straně 46.

Hodnota požadavku *wValue* udává požadovanou hodnotu Latency Timeru v milisekundách.

Parametr	Hodnota
bRequest:	0x09h
wValue:	1 - 255ms
wIndex:	Port
wLength:	0
Data:	none

Tabulka 12: Control Request 0x09h, zdroj: autor

Požadavek nevrací žádná *Data* proto je délka požadavku *wLength* nulová. Hodnota *wIndex* určuje RS232 port, kterého se tento požadavek týká, má smysl pro víceportové obvody, například obvod FT2232D.

3.4.7.8 SIO_GET_LATENCY_TIMER_REQUEST 0x0A

Požadavkem 0x0Ah si ovladač nárokuje navrácení současné hodnoty Latency Timeru obvodu FT232R. Hodnota je v jednotkách milisekund [ms] v rozsahu 1 až 255ms. Skladba USB požadavku je uvedena v tabulce 13 „*Control Request 0x0Ah, zdroj: autor*“ na straně 47.

Parametr	Hodnota
bRequest:	0x0Ah
wValue:	0
wIndex:	Port
wLength:	1
Data:	Actual Latency Timer Value

Tabulka 13: Control Request 0x0Ah, zdroj: autor

Požadavek vrací *Data* odpovídající současné velikosti Latency Timeru obvodu FT232R, délka požadavku *wLength* jsou dva bajty (slovo). Hodnota *wIndex* určuje RS232 port, kterého se tento požadavek týká, má smysl pro víceportové obvody, například obvod FT2232D.

3.4.7.9 SIO_READ_EEPROM_REQUEST 0x90h

Požadavkem 0x90h si ovladač nárokuje navrácení části obsahu vnitřní paměti EEPROM obvodu FT232R. Skladba USB požadavku je uvedena v tabulce 14 „*Control Request 0x90h, zdroj: autor*“ na straně 47.

Parametr	Hodnota
bRequest:	0x90h
wValue:	0
wIndex:	Index pozice v EEPROM
wLength:	Délka dat
Data:	Vrácená data

Tabulka 14: Control Request 0x90h, zdroj: autor

Požadavek vrací *Data* odpovídající obsahu vnitřní EEPROM obvodu FT232R na pozici indexu paměti dané parametrem *wIndex* a v délce *wLength* (v násobcích slov, tedy po dvou bajtech).

Poznámka 3.12 Ikdyž požadavek 0x90h může obecně vrátit jakýkoliv počet bajtů daný parametrem *wLength*, obvod FT232R však žádá vždy jen dva bajty dat, kdy $wLength = 2$.

4 Programové řešení emulátoru

Na základě informací z kapitoly 3 „*Převodník USB-RS232 firmy FTDI*“ na straně 27 a zejména informací z předchozí kapitoly 3.4 „*Převodník typ FT232R programátorský popis*“ na straně 34, byl naprogramován firmware emulátoru převodníku FT232R v mikroprocesoru PIC18F46J50, který bude v této kapitole představen. Ve skutečnosti vznikaly kapitoly současně a provázaně nicméně pro následující výklad a uspořádání kapitol to nemá podstatný význam.

4.1 Základní teorie USB

Teorii USB lze rozdělit na hardwarovou „napájení, USB rezistory, rychlost USB, deskriptory, specifikace tříd zařízení “ a softwarovou „úrovně, rámy, typy přenosů, enumerace, ovladače“.

Některé výše uvedené části USB byly popsány v kapitolách 3.4.2 „*Enumerace*“ na straně 35, 3.4.3 „*Deskriptor*“ na straně 37 a 3.4.4 „*Endpointy*“ na straně 38 a ostatní byly alespoň zmíněny. Pro úplnost je možné všechny informace týkající se USB dohledat v literatuře [12] nebo [27], my se těmito informacemi pro jejich obsáhlost zde podrobněji věnovat nebudeme.

4.2 Základní popis USB frameworku Microchip

USB framework firmy Microchip nazvaný „MCHFSUSB“ jsme krátce představili v kapitole 2.6.6.2 „*Microchip USB framework*“ na straně 25. Zopakujeme, že byl použit USB framework ve verzi v2.9c.

V této kapitole si krátce popíšeme funkce API frameworku MCHFSUSB a to pouze funkce, které byly přímo použity ve firmware pro vytvoření emulátoru. Ostatní API funkce zde popsány nebudou.

4.2.1 USBDeviceInit

Prototyp programové funkce: **void USBDeviceInit()**

Tato funkce musí být volána před jakýmkoliv použitím ostatních USB funkcí včetně funkce *USBDeviceTasks*. Tato funkce inicializuje USB device stack do výchozího stavu a kompletně resetuje hardwarový USB modul mikroprocesoru PIC včetně všech jeho interních proměnných, registrů a příznaků přerušení.

4.2.2 USBDeviceTasks

Prototyp programové funkce: **void USBDeviceTasks()**

Pokud USB stack pracuje v režimu „USB_POLLING“, což je případ našeho firmware, funkce *USBDeviceTasks* by měla být periodicky volána, aby mohly být přijímány a odesílány pakety skrze USB stack. Tato funkce se také stará o přenosy typu (CONTROL_TRANSFER) v průběhu enumerace a také o různé USB události.

Tato funkce by měla být volána každých 1.8ms během procesu enumerace a po jeho skončení minimálně každých 9.8ms nebo častěji, aby nedošlo k přeplnění hardwarového USTAT FIFO mikroprocesoru PIC.

Použití funkce *USBDeviceTasks* a funkce *USBDeviceInit* ilustruje následující výpis kódu 1 „*Použití funkce USBDeviceTasks a funkce USBDeviceInit, zdroj: [10]*“ na straně 50.

```
void main(void) {
    USBDeviceInit();
    while (1) {
        USBDeviceTasks();
        AppTasks();          // Application specific tasks.
    }
}
```

Výpis 1: Použití funkce *USBDeviceTasks* a funkce *USBDeviceInit*, zdroj: [10]

4.2.3 USBEnableEndpoint

Prototyp programové funkce: **void USBEnableEndpoint(BYTE ep, BYTE options)**

Tato funkce povoluje endpoint s definovanými funkcemi. Definované funkce mohou být IN, OUT, HANDSHAKE, SETUP, STALL. Tyto funkce jsou definovány v souboru „usb_hal_pic18.h“ USB frameworku.

Emulátor obvodu FT232R má definovány endpointy v kódu ve výpise 2 „*Definování endpointů emulátoru, zdroj: usb_config.h*“ na straně 50.

```
#define FTDI_DATA_EP_IN 0x01
#define FTDI_DATA_EP_OUT 0x02
#define ENABLE_FTDI_DATA_EP_IN() USBEnableEndpoint(FTDI_DATA_EP_IN, USB_IN_ENABLED |
    USB_HANDSHAKE_ENABLED | USB_DISALLOW_SETUP);
#define ENABLE_FTDI_DATA_EP_OUT() USBEnableEndpoint(FTDI_DATA_EP_OUT,
    USB_OUT_ENABLED | USB_HANDSHAKE_ENABLED | USB_DISALLOW_SETUP);
```

Výpis 2: Definování endpointů emulátoru, zdroj: usb_config.h

4.2.4 USBGetDeviceState

Prototyp programové funkce: **USB_DEVICE_STATE USBGetDeviceState()**

Tato funkce vrací aktuální stav zařízení připojeného ke sběrnici USB. Funkce se používá pro zjištění, zda-li je již zařízení připraveno ke komunikaci po USB sběrnici. Aplikace by se neměla pokoušet posílat nebo přijímat data dokud tato funkce nevrátí (CONFIGURED_STATE).

4.2.5 USBHandleBusy

Prototyp programové funkce: **BOOL USBHandleBusy(USB_HANDLE handle)**

Tato funkce zkontroluje, jestli není definovaný „handle“ zaneprázdněn odesíláním předchozích dat. Nejdříve se musí USB stack postarat o dokončení odeslání předchozích dat, než je možno programově požadovat odeslat další data.

Typický příklad použití je v kódu ve výpisu 3 *„Použití USBHandleBusy, zdroj: [10]“* na straně 51. Zde je „handle“ představován názvem „USBGenericInHandle“. Pokud pod tímto ukazatelem „USBGenericInHandle“ odešleme data a pokusíme se v brzké době odeslat nová bez toho aniž by byly odeslány USB stackem ta předchozí, neumožní nám uvedená část tohoto kódu nová data poslat.

```
//make sure that the last transfer isn't busy by checking the handle
if (!USBHandleBusy(USBGenericInHandle))
{
//Send the data contained in the INPacket[] array out on
// endpoint USBGEN_EP_NUM
USBGenericInHandle = USBGenWrite(USBGEN_EP_NUM,(BYTE*)&INPacket[0],sizeof(INPacket));
}
```

Výpis 3: Použití USBHandleBusy, zdroj: [10]

4.2.6 USBHandleGetLength

Prototyp programové funkce: **WORD USBHandleGetLength(USB_HANDLE handle)**

Tato funkce vrací aktuální délku buferu „handle“. Pokud se funkce použije po ukončení celého přenosu, vrátí celkový počet odeslaných nebo přijatých bajtů.

4.2.7 USBTxOnePacket

Prototyp programové funkce: **USB_HANDLE USBTxOnePacket(BYTE ep, BYTE* data, WORD len)**

Tato funkce pošle data na specifikovaný endpoint a vrátí zpět „handle“, který může být následně funkcí *USBHandleBusy* testován na dokončení operace odeslání dat.

4.2.8 USBRxOnePacket

Prototyp programové funkce: **USB_HANDLE USBRxOnePacket(BYTE ep, BYTE* data, WORD len)**

Tato funkce přijme data ze specifikovaného endpointu a vrátí zpět „handle“, který může být následně funkcí *USBHandleBusy* testován na dokončení operace přijetí dat.

4.2.9 Setup Packet struktura

Setup Packet struktura „SetupPkt“ je hlavní strukturou, která se používá pro vyřizování požadavků ovladače (Requestů). Tato struktura popisuje data obsažená ve standardním USB setup paketu. Tento datový paket je odesílán z počítače pro řízení a konfiguraci USB zařízení. Celý setup paket má velikost 8 bajtů. Struktura „SetupPkt“ je potom definována dle parametrů USB specifikace v kódu ve výpise 4 „*Struktura SetupPkt, zdroj: usb_ch9.h*“ na straně 52.

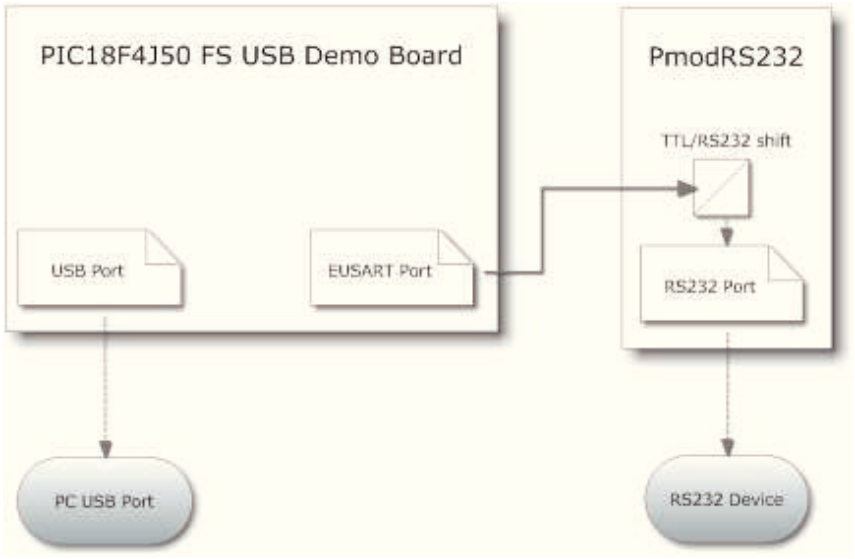
```
struct __attribute__ ((packed))
    {
        BYTE bmRequestType; //from table 9−2 of USB2.0 spec
        BYTE bRequest; //from table 9−2 of USB2.0 spec
        WORD wValue; //from table 9−2 of USB2.0 spec
        WORD wIndex; //from table 9−2 of USB2.0 spec
        WORD wLength; //from table 9−2 of USB2.0 spec
    };
```

Výpis 4: Struktura SetupPkt, zdroj: usb_ch9.h

4.3 Schéma propojení hardware emulátoru

Jak bylo popsáno v kapitole 2.6 „*Popis nástrojů použitých při vývoji*“ na straně 21, emulátor se skládá ze vzájemně propojeného hardware 2.6.1 „*Vývojový kit PIC18F46J50 FS USB Demo Board*“ na straně 21 a 2.6.2 „*Převodník úrovní PmodRS232*“ na straně 22.

Blokové schéma hardwarového propojení je na obrázku 13 „*Blokové schéma propojení komponentů emulátoru, zdroj: autor*“ na straně 53.



Obrázek 13: Blokové schéma propojení komponentů emulátoru, zdroj: autor

Poznámka 4.1 Základní funkční elektronické schéma obou hardwarových desek a jejich vzájemného elektronického propojení je v příloze na obrázku 20 „*Schéma zapojení emulátoru FT232R, zdroj: autor*“ na straně 118.

4.4 Úprava firmware demo CDC

Firmware emulátoru byl postaven na zdrojovém kódu dema „Device - CDC - Serial Emulator“, které je součástí MCHPFSUSB frameworku. Toto demo je vytvořeno nad specifikací USB 2.0 třídy CDC (Communication Device Class), jejíž podrobný popis lze nalézt v referenci [8] a [11].

Tento postup byl zvolen jak z důvodu vývoje emulátoru na vývojovém kitu PIC18F46J50 FS USB Demo Board, tak z důvodu existující funkční struktury USB frameworku se všemi potřebnými soubory a tedy snazší výchozí implementace nového projektu. Celý projekt emulátoru byl vytvořen v novém vývojovém prostředí MPLAB® X IDE.

4.4.1 Popis úpravy

V následujících odstavcích bude krátce popsáno vytvoření projektu emulátoru FTDI USB-RS232 ve vývojovém prostředí MPLAB® X IDE úpravou projektu dema „Device - CDC - Serial Emulator“.

Postup úpravy lze shrnout do následujících bodů:

- přejmenování projektu CDC na FTDI
- uvolnění nepotřebných souborů z projektu
- přidání nových souborů do projektu
- odstranění nepoužitých funkcí a částí kódu
- implementace nových souborů
- nastavení parametrů původních souborů
- konečná očista firmware

Po otevření výchozího demo projektu „Device - CDC - Serial Emulator“ byl projekt přejmenován na „Device - FTDI - Serial Emulator“ a byly z něj uvolněny všechny nepotřebné soubory. Byly uvolněny soubory hlaviček všech hardwarových profilů, ponechán byl pouze harwarový profil pro vývojový kit PIC18F46J50 FS USB Demo Board. Z USB hlaviček byly uvolněny soubory „usb_device_local.h“, „usb_hal.h“, „usb_hal_local.h“ a „usb_hal_pic18.h“. Dále byly uvolněny soubory hlaviček jiných mikroprocesorů než řady PIC18 a soubor hlavičky „usb_function_cdc.h“. Z linker souborů byl ponechán pouze

„rm18F46J50_g.lkr“, ostatní byly taktéž uvolněny. Ze zdrojových USB souborů byl ponechán pouze „usb_device.c“ a „main.c“, všechny ostatní včetně „usb_descriptors.c“ byly uvolněny.

Do projektu byl nově přidán soubor hlavičky „usb_function_ftdi.h“ a k tomu odpovídající zdrojový soubor „usb_function_ftdi.c“. Dále byl přidán nový zdrojový soubor „usb_descriptors_ftdi.c“. Tyto soubory byly zcela nově implementovány.

Nepoužité části kódu, funkce a procedury ve zdrojovém souboru „main.c“ byly odstraněny. Dále musely být nastaveny některé parametry stávajících souborů tak, aby odpovídaly implementovanému emulátoru. Tyto úpravy budou dále popsány. Nakonec byl celý firmware revidován jeho konečnou očistou, byly upraveny komentáře, které vznikaly v průběhu vytváření projektu.

Pro přehlednost je konečná struktura projektu v prostředí MPLAB® X IDE po provedených úpravách uvedena na obrázku 14 „*Struktura projektu emulátoru v prostředí MPLAB® X IDE, zdroj: autor*“ na straně 56.

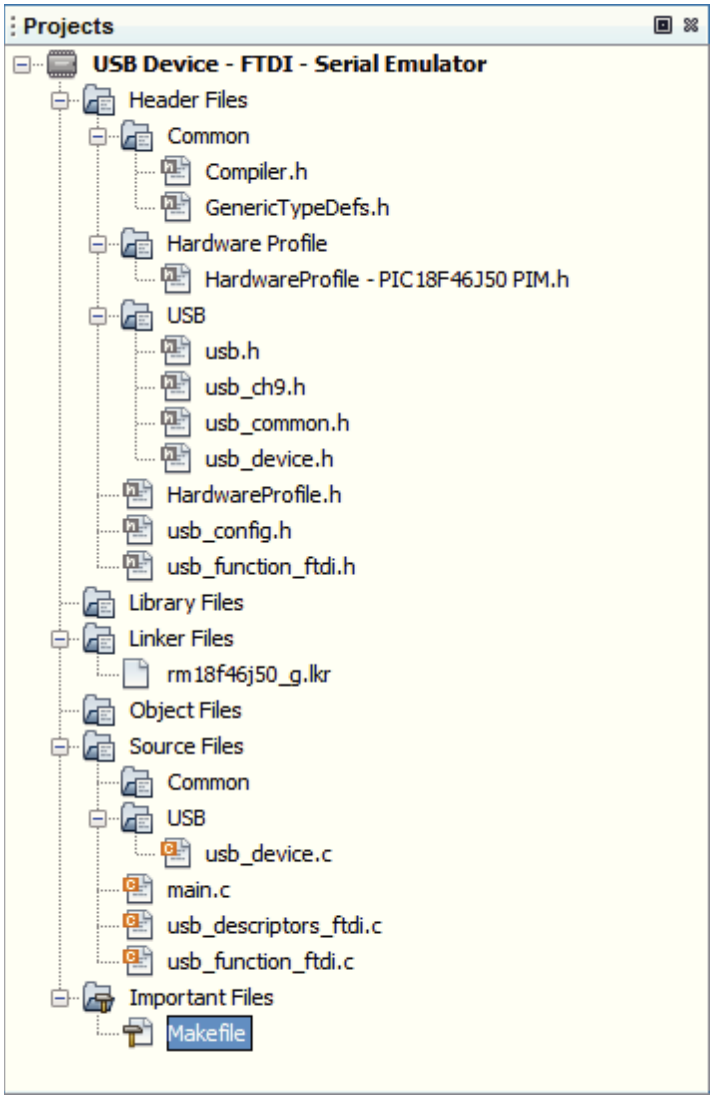
4.5 Realizace firmware emulátoru

V průběhu realizace této práce jsem nenašel žádná konkrétní řešení obdobného úkolu a to ani na jiném typu mikroprocesoru než od výrobce Microchip. Začínal jsem tedy přímo na zelené louce jen s pomocí informací a nástrojů uvedených v kapitole 3.4.1 „*Dostupnost informací*“ na straně 34. V podstatě celá tato kapitola má přímou návaznost na informace poskytnuté v celé kapitole 3 „*Převodník USB-RS232 firmy FTDI*“ na straně 27 a kapitole 4.2 „*Základní popis USB frameworku Microchip*“ na straně 49.

4.5.1 Vlastnosti emulátoru

Do firmware emulátoru byly integrovány následující vlastnosti a funkce emulovaného obvodu FT232R:

- Hardwarové prvky
  - Deskriptor obvodu FT232R
  - Kopie EEPROM originálního FTDI převodníku
  - Softwarový handshaking XON/XOFF
  - Hardwarový handshaking RTS/CTS, DTR/DSR
  - Pravidelné posílání stavu linek a modemu v periodě Latency Timeru
  - Rx/Tx funkce na USB a EUSART port skrze bufery



Obrázek 14: Struktura projektu emulátoru v prostředí MPLAB® X IDE, zdroj: autor

- Softwarové prvky
  - Požadavek ovladače na reset obvodu
  - Ovladačem nastavitelný Latency Timer
  - Ovladačem nastavitelná rychlost komunikace linky RS-232
  - Ovladačem nastavitelné vlastnosti přenosu dat linky RS-232
  - Ovladačem nastavitelná výstupní úroveň CTS/RTS, DSR/DTR pinů
  - Ovladačem dotazovatelný obsah na EEPROM
  - Ovladačem dotazovatelný stav Latency Timeru

- Ovladačem dotazovatelný stav linek a modemu

4.5.2 Součinnost souborů projektu

Pro další popis vývoje emulátoru je nutné mít základní přehled o souborech v projektu a jejich funkci, případně návaznosti na další soubory, protože dále budeme se samotnými soubory konkrétně bez dalšího pracovat.

- USB definiční soubory
  - usb.h (*základní definiční soubor, který vkládá ostatní definiční soubory*)
  - usb_ch9.h (*soubor definic kapitoly 9 specifikace USB 2.0 (USB zařízení)*)
  - usb_common.h (*soubor společných definic MCHPFSUSB framework knihovnu*)
  - usb_device.h (*soubor definic pro USB device zařízení MCHPFSUSB frameworku*)
- USB a HW konfigurační soubory
  - HardwareProfile.h (*automatický výběr kitu na základě použitého mikroprocesouru*)
  - HardwareProfile - PIC18F46J50 PIM.h (*konfigurační soubor vývojového kitu*)
  - usb_config.h (*konfigurační soubor USB device zařízení*)
  - usb_descriptors_ftdi.c (*konfigurační soubor deskriptoru USB device zařízení*)
- Aplikační soubory firmware
  - usb_device.c (*aplikační soubor USB device zařízení MCHPFSUSB frameworku*)
  - main.c (*vstupní bod aplikace, hlavní smyčka firmware*)
  - usb_function_ftdi.c a usb_function_ftdi.h (*aplikační soubor emulátoru FT232R*)

4.5.3 Konfigurace souborů projektu

4.5.3.1 Konfigurace v „HardwareProfile - PIC18F46J50 PIM.h“

Na souborech „HardwareProfile.h“ a „HardwareProfile - PIC18F46J50 PIM.h“ nebylo třeba nic měnit. Soubor „HardwareProfile.h“ správně zvolil použití importu souboru „HardwareProfile - PIC18F46J50 PIM.h“ podle použitého mikroprocesoru PIC18F46J50. Soubor „HardwareProfile - PIC18F46J50 PIM.h“ obsahoval správnou konfiguraci vývojového kitu pro použití USB a jednoho portu EUSART.

4.5.3.2 Konfigurace v „usb_config.h“

Soubor „usb_config.h“ svými direktivami nastavuje chování USB frameworku.

Lze nastavit velikost buferu řídícího endpointu EP0 *USB_EP0_BUFF_SIZE* na hodnoty 8, 16, 32, nebo 64 bajtů, nastavit maximální počet interfejsů v konfiguraci *USB_MAX_NUM_INT* a maximální počet endpointů v konfiguraci *USB_MAX_EP_NUMBER*. Všechny tyto hodnoty slouží USB frameworku pro informaci, kolik paměťového místa v paměti RAM má použít. Správným nastavením lze tedy pamět' šetřit.

Parametrem *USB_NUM_STRING_DESCRIPTORS* se nastavuje počet použitých uživatelských stringů.

Direktiva *USB_SUPPORT_DEVICE* říká USB frameworku, že má zavést při překladu podporu USB device zařízení, tedy že budou použity funkce USB device zařízení a direktiva *USB_POLLING* říká USB frameworku, že bude použit v režimu „polling“ - (opakovaného dotazování).

Dále je zde pasáž, která se věnuje nastavení aliasu endpointů jak budou použity v kódu a aliasu velikosti endpointů, který se použije pro definování velikosti buferu odpovídajícímu endpointu. Samozřejmě tyto informace musí být obdobné jako jsou definovány v deskriptoru v souboru „usb_descriptors_ftdi.c“.

Nastavení uzavírají definice dvou maker *ENABLE_FTDI_DATA_EP_IN()* a *ENABLE_FTDI_DATA_EP_OUT()*, které budou použity v kódu pro inicializaci endpointů v souboru „usb_function_ftdi.c“. Inicializaci endpointů jsme probrali v kapitole 4.2 „*Základní popis USB frameworku Microchip*“ na straně 49 v části 4.2.3 „*USBEnableEndpoint*“ na straně 50.

Nastavení všech parametrů a direktiv je uvedeno ve výpise 5 „*Konfigurace USB frameworku, zdroj: usb_config.h*“ na straně 58.

```
#define USB_EP0_BUFF_SIZE 8
#define USB_MAX_NUM_INT 1
#define USB_MAX_EP_NUMBER 2
#define USB_NUM_STRING_DESCRIPTORS 4
#define USB_SUPPORT_DEVICE
#define USB_POLLING
#define FTDI_DATA_EP_IN 0x01
#define FTDI_DATA_IN_EP_SIZE 0x40
#define FTDI_DATA_EP_OUT 0x02
#define FTDI_DATA_OUT_EP_SIZE 0x40
#define ENABLE_FTDI_DATA_EP_IN() USBEnableEndpoint(FTDI_DATA_EP_IN, USB_IN_ENABLED |
    USB_HANDSHAKE_ENABLED | USB_DISALLOW_SETUP);
#define ENABLE_FTDI_DATA_EP_OUT() USBEnableEndpoint(FTDI_DATA_EP_OUT,
    USB_OUT_ENABLED | USB_HANDSHAKE_ENABLED | USB_DISALLOW_SETUP);
```

Výpis 5: Konfigurace USB frameworku, zdroj: usb_config.h

Poznámka 4.2 V direktivě definování aliasu endpointu je použito prosté číslo endpointu, nikoliv jeho fyzická adresa, která definuje také směr toku dat daného endpointu.

4.5.3.3 Konfigurace v „usb_descriptors_ftdi.c“

V souboru „usb_descriptors_ftdi.c“ je definován USB deskriptor zařízení tak, aby se emulátor tvářil jako originální obvod FT232R. USB deskriptor zařízení vlastní jeden konfigurační deskriptor s jedním interfejsem se dvěma endpointy a nakonec čtyři uživatelské string řetězce.

Deskriptor zařízení má svojí vlastní strukturu *USB_DEVICE_DESCRIPTOR* definovanou v souboru „usb_ch9.h“, která byla naplněna pod jménem *device_dsc*.

Konfigurační deskriptor je uložen do pole *USB_CD_Ptr[]* a uživatelské string řetězce do pole *USB_SD_Ptr[USB_NUM_STRING_DESCRIPTORS]*.

Se všemi těmito daty následně pracuje zdrojový soubor „usb_device.c“. Nelze opomenout, že každá část deskriptoru má svoji délku, kterou je třeba spočítat a uvézt na příslušném místě deskriptoru. Tato informace je zmíněna v komentáři zdrojového kódu.

Obvod FT232R používá, kromě standardního řídícího endopintu EP0, také endpoint EP01 IN pro přenos dat z obvodu FT232R do USB a endpoint EP02 OUT pro přenos dat z USB do obvodu FT232R. Oba tyto endpointy jsou v deskriptoru definovány svojí skutečnou adresou, která je složena z čísla endpointu a jeho funkce buď jako vstupu nebo výstupu. Podrobné informace lze nalézt v USB specifikaci.

Protože je deskriptor specifický pro jakoukoliv funkčnost USB zařízení a jeho prvotní enumeraci, umístil jsem celý výpis kódu zvlášť do přílohy 24 „*Deskriptor emulátoru, zdroj: usb_descriptors_ftdi.c*“ na straně 110.

Poznámka 4.3 Parametr *USB_NUM_STRING_DESCRIPTORS* se nastavuje v konfiguračním souboru „usb_config.h“ a určuje počet použitých uživatelských stringů, v našem případě čtyři.

Poznámka 4.4 Hodnoty deskriptoru bDeviceClass, bDeviceSubClass a bDeviceProtocol používá operační systém k nalezení ovladače pro danou třídu USB zařízení. Pokud jsou tyto hodnoty nulové, jako v případě obvodu FT232R, je zařízení ohlášeno jako typu *VENDOR* a vlastní popis zařízení a nalezení ovladače se děje až přes interfejs konfiguračního deskriptoru.

Poznámka 4.5 Soubor „usb_descriptors_ftdi.c“ byl postaven na původním souboru „usb_descriptors.c“, protože jeho struktura je vázána na USB framework. Zejména pak soubor „usb_device.c“, používá tento konfigurační deskriptor ke své činnosti, převážně pro enumeraci. Proto byl v souboru „usb_descriptors_ftdi.c“ ponechán také licenční ko-

mentář výrobce Microchip, který detailněji rozvádí strukturu souboru deskriptoru a správné definování uživatelských stringů zařízení.

4.5.3.4 Konfigurace v „main.c“

V souboru „main.c“ je konfigurace použitého mikroprocesoru PIC18F46J50. Kromě ostatního nastavení mikroprocesoru je pro projekt emulátoru nejdůležitější konfigurace oscilátoru, kterou si zde krátce za použití informací uvedených v kapitole 2.5.5 *„Dostupné konfigurace oscilátoru“* na straně 16 konkrétně popíšeme.

Vývojový kit používá externí krystal 12MHz, jak je patrno ze schematu zapojení na obrázku 20 *„Schéma zapojení emulátoru FT232R, zdroj: autor“* na straně 118. Pro práci USB zařízení ve Full-Speed režimu musí být do vnitřního USB modulu mikroprocesoru PIC18F46J50 přivedena frekvence 48MHz.

Protože používáme externí krystal a pro potřebnou frekvenci USB budeme potřebovat interní PLL, nastavíme oscilátor do režimu HSPLL. Protože vstupní kmitočet do PLL musí být přes jeho děličku přesně 4MHz (ze kterého PLL na výstupu vyrobí 96MHz), potřebujeme při vstupním kmitočtu krystalu 12MHz dělit 3. Nastavíme tedy děličku PLL na 3. Poslední direktivou konfigurace oscilátoru nastavíme interní frekvenci na shodnou s USB modulem (48MHz) nastavením CPU děličky na poměr 1:1.

Poznámka 4.6 Konfigurace mikroprocesoru se provádí skrze konfigurační bity konfiguračních registrů. Tyto bity se nastavují při programování mikroprocesoru a zůstávají nastaveny až do jeho přeprogramování. Kompletní informace ke konfiguračním bitům a konfiguračním registrům mikroprocesoru jsou podrobně popsány v katalogovém listu [6] výrobce.

Poznámka 4.7 V praxi se neprovádí nastavování jednotlivých konfiguračních bitů mikroprocesoru ručně, ale používají se direktivy kompilátoru *#pragma config*, které automaticky nastaví příslušné bity podle vybrané direktivy a její funkce. Každý mikroprocesor má vlastní set konfiguračních direktiv *#pragma config* pro svůj kompilátor. Konfigurační direktivy mikroprocesorů kompilátoru PIC C18 lze nalézt po nainstalování kompilátoru v souboru „hlpPIC18ConfigSet.chm“.

Použití direktiv pro nastavení konfigurace mikroprocesoru emulátoru FT232R je znázorněno v kódu ve výpisu 6 *„Nastavení konfigurace mikroprocesoru PIC18F46J50, zdroj: main.c“* na straně 61.

```
#pragma config OSC = HSPLL      //HS oscillator, PLL enabled, HSPLL used by USB
#pragma config PLLDIV = 3       //Divide by 3 (12 MHz oscillator input)
#pragma config CPUDIV = OSC1    //No CPU system clock divide

#pragma config XINST = OFF      //Extended instruction set disabled
#pragma config CP0 = OFF        //Program memory is not code-protected
#pragma config WDTEN = OFF      //WDT disabled (enabled by SWDTEN bit)
#pragma config STVREN = ON        //stack overflow/underflow reset enabled
#pragma config FCMEN = OFF      //Fail-Safe Clock Monitor disabled
#pragma config IESO = OFF       //Two-Speed Start-up disabled
#pragma config WDTPS = 32768    //1:32768
#pragma config DSWDTOSC = INTOSCREF //DSWDT uses INTOSC/INTRC as clock
#pragma config RTCOSC = T1OSCREF //RTCC uses T1OSC/T1CKI as clock
#pragma config DSBOREN = OFF    //Zero-Power BOR disabled in Deep Sleep
#pragma config DSWDTEN = OFF    //Disabled
#pragma config DSWDTPS = 8192   //1:8,192 (8.5 seconds)
#pragma config IOL1WAY = OFF    //IOLOCK bit can be set and cleared
#pragma config MSSP7B_EN = MSK7 //7 Bit address masking
#pragma config WPFP = PAGE_1    //Write Protect Program Flash Page 0
#pragma config WPEND = PAGE_0   //Start protection at page 0
#pragma config WPCFG = OFF      //Write/Erase last page protect Disabled
#pragma config WPDIS = OFF      //WPFP[5:0], WPEND, and WPCFG bits ignored
```

Výpis 6: Nastavení konfigurace mikroprocesoru PIC18F46J50, zdroj: main.c

4.5.4 Blokové schéma firmware emulátoru

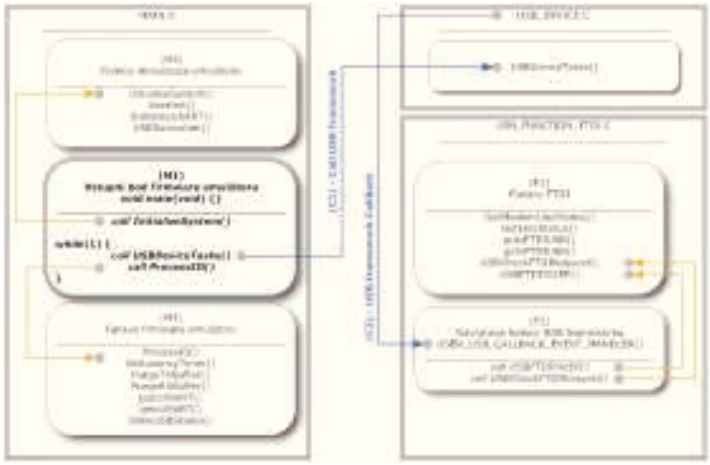
V této kapitole si popíšeme bloky jednotlivých částí kódu zdrojových souborů, které spolu logicky souvisí. Zejména půjde o spolupráci mezi zdrojovými soubory „main.c“, „usb_function_ftdi.c“ a USB frameworkem zastoupeným představitelem „usb_device.c“. Jednak tuto informaci potřebujeme pro pochopení širších souvislostí projektu emulátoru a jednak se k tomuto nebudeme muset v dalších částech textu vícekrát podrobněji vracet. V následujícím textu si potom vysvětlíme některé důležité funkce, které vznikají součinností některých bloků a následně se budeme podrobněji věnovat jednotlivým důležitým částem kódu nebo jednotlivým funkcím.

Schématické znázornění bloků jednotlivých částí kódu naleznete na obrázku 15 „*Blokový diagram firmware emulátoru, zdroj: autor*“ na straně 62, jehož popis bude následovat.

Poznámka 4.8 Blokové schéma firmware emulátoru ve větší velikosti naleznete v příloze na obrázku 21 „*Blokový diagram firmware emulátoru, zdroj: autor*“ na straně 120.

Pro snažší orientaci mezi popisným textem a schématickým znázorněním byly jednotlivé části ve schématu označeny následujícími symboly:

- (M1) - Hlavní smyčka firmware



Obrázek 15: Blokový diagram firmware emulátoru, zdroj: autor

- (M2) - Inicializace firmware emulátoru
- (M3) - Hlavní funkce emulátoru
- (F1) - Funkce FTDI
- (F2) - Návratová Funkce USB frameworku
- (C1) - Call USB framework
- (C2) - USB framework CallBack

Uvedené symboly jsou obsaženy v názvu každé kapitoly odkazující na popis ve schématu. Písmeno M značí odkaz na blok kódu ve zdrojovém souboru Main, písmeno F značí odkaz na blok kódu ve zdrojovém souboru funkce FTDI a písmeno C označuje cestu volání USB frameworku. Každé písmeno má přiřazenu pořadovou číslovku, kterou se odlišuje ve své vlastní skupině.

4.5.4.1 Hlavní smyčka firmware (M1)

Firmware emulátoru běží v nekonečné smyčce *while(1)* souboru *main.c*. Než se do ní dostane, provede se nejdříve inicializace emulátoru programovou funkcí *InitializeSystem*, která bude popsána níže v části 4.5.4.3 „*Hlavní funkce emulátoru (M3)*“ na straně 63. Ve smyčce se potom opakovaně provádí volání úloh USB frameworku, běží tzv. „pooling“ režim, který byl nastaven v konfiguraci uvedené v kapitole 4.5.3.2 „*Konfigurace v „usb_config.h““* na straně 57. Volání úloh frameworku se musí opakovat s periodou maximálně 1.8ms nebo častěji. Volání USB frameworku je naznačeno spojnicí *Call USB framework 1.8ms* ve schematu na obrázku 15 „*Blokový diagram firmware emulátoru, zdroj: autor*“ na straně

62. Úlohy USB frameworku se volají programovou funkcí *USBDeviceTasks* a trvání této funkce není zpravidla delší než 100 instrukčních cyklů. Kód hlavní smyčky je na výpise programu 7 „*Hlavní programová smyčka emulátoru, zdroj: main.c*“ na straně 63.

```
void main(void) {

InitializeSystem () ;
   while (1) {
      USBDeviceTasks(); // takes 100instruction cycles
      ProcessIO();  // Application related code and tasks.
   }
}//end main
```

Výpis 7: Hlavní programová smyčka emulátoru, zdroj: main.c

4.5.4.2 Inicializace firmware emulátoru (M2)

Programová funkce *InitializeSystem* provede inicializaci PLL mikroprocesoru a inicializací jeho I/O linek a následně zavolá funkce inicializace *UserInit* a *USBDeviceInit*.

Funkce inicializace *UserInit* se postará o inicializaci buferů, inicializaci časovače TIMER0 použitého pro Latency Timer a inicializaci diod LED vývojového kitu a následně zavolá funkce *InitializeUSART* a *SetLatencyTimer*.

Programová funkce *InitializeUSART* se postará o nastavení výstupních pinů modulu EUSART, nastavení výchozí rychlosti linky RS232 na 19200 baud a provede se inicializace řídících linek portu RS232 RTS, CTS, DTS a DTR. Programová funkce *SetLatencyTimer* nastaví Latency Timer do výchozí hodnoty 16ms.

Posledně volaná funkce *USBDeviceInit* je funkcí USB frameworku, která musí být volána před jeho prvním použitím a před použitím funkce *USBDeviceTasks*.

4.5.4.3 Hlavní funkce emulátoru (M3)

Hlavní funkcí emulátoru, která vykonává celou činnost převodníku USB-RS232, je funkce *ProcessIO*. Vzhledem ke své důležitosti a vnitřní logice bude tato funkce popsána v samostatné části 4.5.5 „*ProcessIO*“ na straně 66.

Dalšími funkcemi jsou funkce pro odeslání a přijetí znaku na EUSART portu mikroprocesoru PIC18F46J50. Funkce *putcUSART* zabezpečuje odeslání znaku na port RS232 a funkce *getcUSART* zabezpečuje přijetí znaku odeslaného vnějším zařízením na Rx port modulu EUSART. Pro zpracování požadavku ovladače na vyčištění buferu jsou implementovány příslušné funkce *PurgeTXBuffer* a *PurgeRXBuffer*. Všechny tyto funkce jsou volány podle potřeby.

Poslední programovou funkcí je *BlinkUSBStatus*. Tato funkce je původní převzatou funkcí dema CDC pro vývojový kit PIC18F46J50 FS USB Demo Board. Tato funkce zabezpečuje blikání dvou LED diod, kterými je vývojový kit vybaven v závislosti na stavu firmware. Potom co je USB zařízení rozpoznáno blikají střídavě obě led s periodou přibližně 0.5 sekund.

4.5.4.4 Funkce FTDI (F1)

Funkce *putsFTDIUSB* zabezpečují poslání znaku na USB port odesláním jednoho USB packetu. Funkce *getsFTDIUSB* zabezpečuje přijetí znaku z USB portu přijetím jednoho USB packetu. Obě funkce pracují s buferem, lze tedy odesílat a posílat více znaků najednou.

Funkce *getsFTDIUSB* pracuje s buferem *RS232_Out_Data*, který plní přijatými znaky z USB portu, které jsou následně odesílány na Tx port EUSART. Tímto se zabezpečuje přenos dat z portu USB na port RS232.

Funkce *putsFTDIUSB* pracuje s buferem *USB_Out_Buffer*, který je plněn přijatými znaky z RS232 portu, a které jsou následně odesílány touto funkcí na USB port. Tímto se zabezpečuje přenos dat z portu RS232 na port USB.

Funkce *GetLineStatus* a *GetModemLineStatus* vrací status CTS a DTS linek portu RS232 a status modemu emulovaného obvodu FT232R. Funkce jsou volány dle potřeby hlavní funkce emulátoru nebo na základě požadavku USB ovladače.

Funkce *USBFTDIInitEP* je volána jen jednou ihned po úspěšném rozpoznání zařízení emulátoru po připojení k USB portu. Funkce má na starosti nastavení výchozí komunikační rychlosti portu RS232 převodníku, povolení všech registrovaných endpointů zařízení, výchozí inicializaci handle Tx USB packetu a výchozí inicializaci handle Rx USB packetu přijetím prvního packetu z USB zařízení.

Funkce *USBCheckFTDIRequest* je natolik specifická a důležitá, že bude popsána v samostatné části 4.5.6 „*Zpracování požadavků FTDI ovladače*“ na straně 68.

4.5.4.5 Návratová Funkce USB frameworku (F2)

Návratová funkce USB stacku frameworku *USER_USB_CALLBACK_EVENT_HANDLER()* je volána přímo z USB frameworku, aby dala na vědomí uživatelské aplikaci novou USB událost, která nastala.

Uživatelská aplikace, v našem případě firmware emulátoru převodníku USB-RS232 slyší celkem na dvě událsoti USB frameworku. První událostí je úspěšná enumerace zařízení emulátoru po připojení k USB portu, pak je volána funkce *USBFTDIInitEP*, kterou jsme si popsali ve stati 4.5.4.4 „*Funkce FTDI (F1)*“ na straně 64. Druhou událostí je jakýkoliv

požadavek ovladače na řídícím kanálu USB zařízení tedy na endpointu EP0. Všechny tyto požadavky je třeba zpracovat. Samotné zpracování provádí programová funkce *USBCheckFTDIRequest*, jíž se budeme samostatně věnovat v kapitole 4.5.6 „*Zpracování požadavků FTDI ovladače*“ na straně 68.

Výpis návratové funkce USB stack frameworku *USER_USB_CALLBACK_EVENT_HANDLER()* je výpise kódu 8 „*Návratová Funkce USB frameworku, zdroj: usb_function_ftdi.c*“ na straně 65.

```
BOOL USER_USB_CALLBACK_EVENT_HANDLER(USB_EVENT event, void *pdata, WORD size) {
    switch (event) {
        case EVENT_TRANSFER:
            break;
        case EVENT_SOF:
            break;
        case EVENT_SUSPEND:
            break;
        case EVENT_RESUME:
            break;
        case EVENT_CONFIGURED:
            USBFTDIInitEP();
            break;
        case EVENT_SET_DESCRIPTOR:
            break;
        case EVENT_EP0_REQUEST:
            USBCheckFTDIRequest();
            break;
        case EVENT_BUS_ERROR:
            break;
        case EVENT_TRANSFER_TERMINATED:
            break;
        default:
            break;
    }
    return TRUE;
}
```

Výpis 8: Návratová Funkce USB frameworku, zdroj: usb_function_ftdi.c

4.5.4.6 Call USB framework (C1)

Spojnicí *Call USB framework (C1)* je znázorněno periodické volání funkce *USBDeviceTasks*, která byla posána v části 4.2.2 „*USBDeviceTasks*“ na straně 50. Zopakujme si zde, že funkci je nutno volat periodicky nejpozději každých 1.8ms.

4.5.4.7 USB framework CallBack (C2)

Spojnicí *USB framework CallBack (C2)* je znázorněno návratové volání funkce *USER_USB_CALLBACK_EVENT_HANDLER()* z USB stacku frameworku. Funkce byla posána v části 4.5.4.5 „*Návratová Funkce USB frameworku (F2)*“ na straně 64. Zopakujme si zde, že funkce dává na vědomí uživatelské aplikaci novou USB událost, která nastala.

4.5.5 ProcessIO

Programová funkce *ProcessIO* zabezpečuje celou logiku emulátoru převodníku USB-RS232. Stará se o přijetí dat z USB portu a přijetí dat z EUSART RS232 portu. Přijatá data funkce ukládá do interních buferů odkud je následně zase odesílá na opačný port než odkud data přišla.

Znázornění vývojového diagramu naleznete na obrázku 16 „*Vývojový diagram funkce ProcessIO firmware emulátoru, zdroj: autor*“ na straně 67.

Poznámka 4.9 Znázornění vývojového diagramu firmware emulátoru ve větší velikosti naleznete v příloze na obrázku 22 „*Vývojový diagram funkce ProcessIO firmware emulátoru, zdroj: autor*“ na straně 121.

Vývojový diagram si nyní detailněji popíšeme a budeme se odvolávat na jeho části. Ještě si zde připomeňme práci dvou paměťových míst „buferů“, které programová funkce *ProcessIO* používá ke své práci a na které se budeme v textu níže odvolávat. Přijatá data z portu USB jsou ukládána do buferu *RS232_Out_Data* určeného pro odeslání dat na port RS232, odkud budou pravidelně vyzvedána. Přijatá data z portu RS232 jsou ukládána do buferu *USB_Out_Data* určeného pro odeslání dat na port USB, odkud budou pravidelně vyzvedána, viz. vývojový diagram.

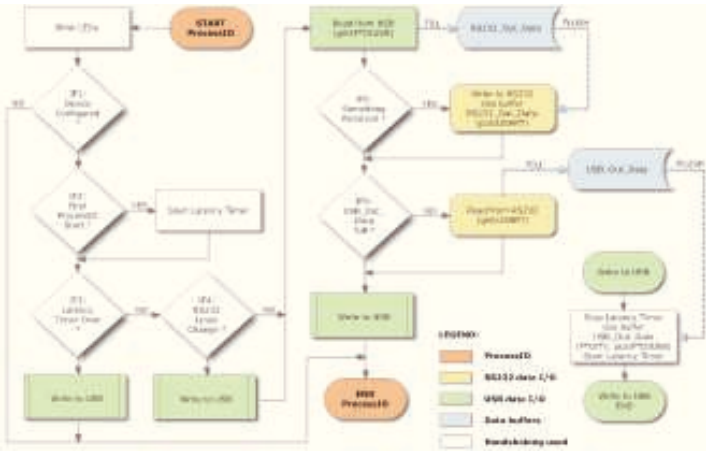
Práce programové funkce *ProcessIO* začíná ve chvíli, kdy bylo zařízení emulátoru rozpoznáno operačním systémem a úspěšně enumerováno. Protože je funkce umístěna v hlavní smyčce firmware, která se provádí od chvíle připojení emulátoru na napájení a tedy v době, kdy zařízení není ještě enumerováno, musí být provedení funkce *ProcessIO* po dobu než dojde k enumeraci zastaveno. Úspěšná enumerace zařízení je tedy zkontrolována ihned po zavolání funkce testovou podmínkou *IF1:* jako první kód v jejím těle.

V případě, že zařízení emulátoru nebylo doposud enumerováno, provádění programové funkce *ProcessIO* je předčasně ukončeno.

Nyní začíná pracovat smyčka Latency Timeru, kterou jsme si vysvětlili v kapitole 3.4.6 „*Latency Timer*“ na straně 39. Jak je tento Latency Timer aplikován ve firmware, bude představeno v kapitole 4.5.8.1 „*Výpočet pro nastavení Latency Timeru*“ na straně 73.

Latency Timer zabezpečuje, že jsou na port USB v pravidelném čase nastaveném hodnotou Latency Timeru odesílaná data o stavu linek portu RS232, o stavu modemu tohoto portu a pokud v daném čase existují, tak také data z portu RS232 přijatá. Aby mohl Latency Timer správně pracovat, je nejdříve nutno jej po enumeraci zařízení spustit. Tento start zabezpečuje testová podmínka *IF2:*, která vyhodnocuje první spuštění programové funkce *ProcessIO* a v případě úspěšnosti testu spustí časovač Latency Timer. V dalším zpracování při běhu firmware je podmínka vždy vyhodnocena jako neúspěšná a první start časovače Latency Timer již není proveden.

Následující podmínka *IF3:* už je pravým jádrem programové funkce *ProcessIO*, která zajišťuje automatické posílání dat na port USB. Podmínka *IF3:* testuje čas nastavený pro Latency Timer a pokud tento čas vypršel, je ihned provedeno makro pro zápis dat na portu USB. Makro nejdříve zabezpečí stopnutí časovače Latency Timeru, následně je provedeno odeslání dat na port USB a nakonec je časovač Latency Timer znovu inicializován a spuštěn. Tímto dochází k odesílání dat na USB port v nekonečném koloběhu stanovenému časem Latency Timer časovače.



Obrázek 16: Vývojový diagram funkce ProcessIO firmware emulátoru, zdroj: autor

Druhou částí podmínky *IF3:* je její vyhodnocení, kdy časovač Latency Timer doposud nevypršel. V tomto případě provádí programová funkce svoji druhou neméně důležitou úlohu a to plnění buferů pro odesílání dat a činností s tímto spojených.

První takovou činností je kontrola stavu linek CTS, DSR portu RS232. V případě, že se změní stav některé z těchto linek, musí o tom být ovladač USB zařízení ihned informován, protože tyto linky portu RS232 mohou být ovladačem použity pro hardwarové řízení handshakingu. V případě nastaveného handshakingu provádí testování linek podmínka *IF4:* a v případě, že je podmínka úspěšná, provede se ihned odeslání informace o stavu linek na port USB.

Následující činností už je samotný převod dat mezi porty USB a RS232. Logika je rozdělena do dvou kroků.

V prvním kroku se zabezpečuje přesun dat z portu USB na port RS232. Z USB portu se přečtou data a uloží do buferu *RS232_Out_Data*. V případě, že je úspěšná podmínka *IF5:*, která testuje, zda-li byly nějaká data z portu USB přijata, jsou tyto data následně z buferu vybrána a ihned přenesena na port RS232.

Ve druhém kroku se zabezpečuje přesun dat z portu RS232 na port USB. Podmínka *IF6:* testuje naplněnost buferu *USB_Out_Data*. V případě, že není naplněn, přečte se další znak z portu RS232 a uloží do tohoto buferu. Pokud je však bufer již naplněn, uložení dalšího znaku do tohoto buferu není možné a musí být nejdříve bufer *USB_Out_Data* vyprázdněn okamžitým odesláním jeho obsahu na port USB.

Tímto práce programové funkce *ProcessIO* končí. Pro úplnost ještě uvedu, že ve vývojovém diagramu jsou barevně zakresleny logicky spolu související části. Zeleně je označeno zpracování dat na portu USB a oranžově zpracování dat na portu RS232. Modře jsou vyznačeny bufery pro USB a RS232, přerušovanými šipkami k nim vedoucími je naznačeno plnění buferů daty a přerušovanými šipkami vedoucími z těchto buferů je naznačeno odebírání dat z těchto buferů. Nakonec červeně je zakreslen vstup a výstup do programové funkce *ProcessIO*. Barevná legenda je obsahem vývojového diagramu.

Programová funkce *ProcessIO* řídí také softwarový a hardwarový handshaking. Uvedený vývojový diagram se tímto však nikterak nemění, protože hw handshaking je implementován uvnitř bloků vývojového diagramu. Podrobnější práce handshakingu emulátoru převodníku bude pospána v kapitole 4.5.7 „*Handshaking*“ na straně 69.

4.5.6 Zpracování požadavků FTDI ovladače

Zpracování požadavků přicházejících po řídícím kanálu endpointu EP0 provádí, jak jsme se již informovali, funkce *USBCheckFTDIRequest*. Tato funkce zpracovává všechny požadavky, které byly představeny v popisu obvodu FT232R v kapitole 3.4.7 „*Popis USB protokolu FTDI - Control Requests*“ na straně 40.

Není zde namístě znovu opisovat jednotlivé činnosti požadavků a popisovat jejich implementaci ve firmware emulátoru. Implementace byla přímá dle představené spe-

cifikace, nebyla složitá a je dostupná ve zdrojových kódech projektu včetně krátkých komentářů.

Obecně jen uvedeme, že ze všech požadavků přicházejících po endpointu EP0 se vybírají pouze ty, které jsou určeny pro zařízení typu *VENDOR*. Jak jsme si pospali v kapitole 4.5.3.3 „*Konfigurace v „usb_descriptors_ftdi.c““* na straně 58 typ zařízení je stanoven jeho deskriptorem.

V kódu se tedy filtrují pouze požadavky, jejichž typ je *USB_SETUP_TYPE_VENDOR_BITFIELD*. Funkční výpis kódu zprácovávaných požadavků bez jednotlivých implementací je ve výpise kódu 9 „*Náhled funkce USBCheckFTDIRequest, zdroj: usb_function_ftdi.c*“ na straně 69.

```
void USBCheckFTDIRequest() {
    if  (SetupPkt.RequestType != USB_SETUP_TYPE_VENDOR_BITFIELD) return;
    switch (SetupPkt.bRequest) {
        case SIO_RESET_REQUEST:
            break;
        case SIO_SET_MODEM_CTRL_REQUEST:
            break;
        case SIO_SET_FLOW_CTRL_REQUEST:
            break;
        case SIO_SET_BAUDRATE_REQUEST:
            break;
        case SIO_SET_DATA_REQUEST:
            break;
        case SIO_POLL_MODEM_STATUS_REQUEST:
            break;
        case SIO_SET_LATENCY_TIMER_REQUEST:
            break;
        case SIO_GET_LATENCY_TIMER_REQUEST:
            break;
        case SIO_READ_EEPROM_REQUEST:
            break;
        default:
            break;
    }
}
```

Výpis 9: Náhled funkce *USBCheckFTDIRequest*, zdroj: usb_function_ftdi.c

Poznámka 4.10 Jak je třeba nastavit modul EUSART mikroporcesoru PIC18F46J50 na požadovaný baudrate bude popsáno v kapitole 4.5.8.2 „*Výpočet pro nastavení baudrate*“ na straně 75. Požadavek *SIO_SET_BAUDRATE_REQUEST* byl teoreticky rozebrán v kapitole 3.4.7.4 „*SIO_SET_BAUDRATE_REQUEST 0x03h*“ na straně 43.

4.5.7 Handshaking

Činnost handshakingu byla implementována podle originálního FTDI převodníku, což bylo ověřeno porovnáním práce originálního převodníku a jeho emulátoru. Tato činnost spočívá v tom, že handshaking na portu RS232 si řídí samotný obvod FT232R. Výsledkem je, že pokud obdrží převodník požadavek z portu RS232 na stopnutí dodávky dat, nečeká na signál z ovladače a stopne odesílání dat na port Tx RS232 okamžitě. Směr USB na RS232 zůstává otevřen a veškerá data převodník uchovává v buferu. Po zrušení požadavku na stopnutí dodávky dat, převodník nepřerušeně odešle z buferu na pin Tx portu RS232 mezitím uchované data. Toto chování lze vysledovat přes hardwarový loopback odpojením a následným připojením propojky RTS/CTS v průběhu psaní textu v terminálu.

Funkci interního handshakingu na úrovni emulátoru, jak byla pospána v odstavci výše, implementuje programová funkce *ProcessIO*. Funkce handshakingu na úrovni ovladače je implementována ve zdrojovém souboru „usb_function_ftdi.c“.

Blokový diagram implementovaného handshakingu je na obrázku 17 „*Vývojový diagram handshakingu emulátoru, zdroj: autor*“ na straně 70. Na diagram se budeme v textu níže dále odvolávat.

V blokovém diagramu je zakreslena cesta dat mezi portem USB a portem RS232 a řízení provozu těchto dat pomocí handshakingu. Zeleně je označena cesta, kterou ovladač prostřednictvím USB portu dává na vědomí požadavek na změni linek DTR a RTS portu RS232 a naopak, kdy pravidelnou kontrolou emulátor zjišťuje stav linek DSR a CTS a informuje o tomto stavu ovladač odesláním této informace na USB port. Žlutě je potom zakreslena činnost emulátoru, který sleduje linky DSR a CTS portu RS232 a sleduje XON/XOFF znak v příchozích datech z pinu Rx portu RS232 a na základě této informace rozhoduje o pozastavení a uvolnění posílání dat na pin Tx portu RS232, což je v diagramu vyznačeno přerušovanou červenou čarou. Použité barevné schéma je součástí legendy blokového diagramu.

Obrázek 17: Vývojový diagram handshakingu emulátoru, zdroj: autor

Na obě části, handshakingu na úrovni emulátoru a handshakingu na úrovni ovladače, se níže v textu krátce podíváme.

Poznámka 4.11 Blokový diagram implementovaného handshakingu v plném rozlišení naleznete v příloze na obrázku 23 „*Vývojový diagram handshakingu emulátoru, zdroj: autor*“ na straně 122.

4.5.7.1 Handshaking na úrovni emulátoru

Handshaking na úrovni emulátoru implementuje programová funkce *ProcessIO*. Její blokové schéma je zakresleno ve vývojovém diagramu na obrázku 16 „*Vývojový diagram funkce ProcessIO firmware emulátoru, zdroj: autor*“ na straně 67. Bloky, které obsahují funkce handshakingu, jsou označeny obdélníkem se zakulacenými rohy. Jak je patrno z diagramu, skutečně je funkce handshakingu použita pouze v součinnosti s portem RS232 a to ve dvou podmínkách *IF5:* a *IF6:*.

Kód v podmínce *IF5:* kontroluje, zda-li byla přijata data z USB portu a pokud ano, je jejím úkolem data přeposlat na port RS232. Pokud je však povolený handshaking, musí kód nejdříve zkontrolovat, zda-li je odeslání oprávněné. V případě, že handshaking v danou chvíli zakazuje odeslání dat, nebudou na port RS232 odeslány.

Oprávněnost odeslání kontroluje firmware přímým dotazem na stav RS232 linek v případě hardwarového handshakingu a v případě softwarového handshakingu kontroluje flag „USART_Transmit_enabled“, jehož stav mu připravuje kód v podmínce *IF6:*, kterou si vysvětlíme níže. Funkční část tohoto kódu je zobrazena ve výpise 10 „*HW handshaking podmínky IF5:, zdroj: main.c*“ na straně 71.

```
// IF5:
if (RS232_Out_Data_Rdy && flagTxUSARTReady()) {
    if (FlowControlFlag.RTSCTS == 1) {
        if (UART_CTS == CTS_ACTIVE_LEVEL ^ 1) {
            USARTTx();
        }
    } else if (FlowControlFlag.DTRDSR == 1) {
        if (UART_DSR == DSR_ACTIVE_LEVEL ^ 1) {
            USARTTx();
        }
    } else if (FlowControlFlag.XonXoff == 1) {
        if (USART_Transmit_enabled == TRUE) {
            USARTTx();
        }
    } else {
        USARTTx();
    }
}
```

Výpis 10: HW handshaking podmínky IF5:, zdroj: main.c

Kód v podmínce *IF6:* zajišťuje čtení znaku z portu RS232 a v případě, že je povolen softwarový handshaking, porovnává příchozí znak oproti znakům definovaným pro funkci XON a XOFF. V případě odhalení tohoto znaku nastavuje kód flag „USART_Transmit_enabled" do příslušného stavu „XOFF: USART_Transmit_enabled = FALSE" a „XON: USART_Transmit_enabled = TRUE". Tato informace je následně zpracována v podmínce *IF5:* pro zablokování nebo povolení přenosu dat na linku Tx portu RS232. Funkční část kódu je zobrazena ve výpise 11 „*HW handshaking podmínky IF6:, zdroj: main.c*" na straně 71.

```
// IF6:
if (FlowControlFlag.XonXoff == 1) {
   c = getcUSART();
   if (c == XOFF) {
      USART_Transmit_enabled = FALSE;
      USB_Out_Data[NextUSBOut] = c;
      ++NextUSBOut;
      FTDITx();
   } else if (c == XON) {
      USART_Transmit_enabled = TRUE;
      USB_Out_Data[NextUSBOut] = c;
      ++NextUSBOut;
   } else {
      USB_Out_Data[NextUSBOut] = c;
      ++NextUSBOut;
   }
} else {
   c = getcUSART();
   USB_Out_Data[NextUSBOut] = c;
   ++NextUSBOut;
}
```

Výpis 11: HW handshaking podmínky IF6:, zdroj: main.c

```
case SIO_SET_MODEM_CTRL_REQUEST:
   switch (SetupPkt.wValue) {
   case 0x0101:
      UART_DTR = (DTR_ACTIVE_LEVEL ^ 1);
      break;
   case 0x0100:
      UART_DTR = DTR_ACTIVE_LEVEL;
      break;
   case 0x0202:
      UART_RTS = (RTS_ACTIVE_LEVEL ^ 1);
      break;
   case 0x0200:
      UART_RTS = RTS_ACTIVE_LEVEL;
      break;
   default:
      break;
}
```

Výpis 12: HW handshaking ovladače, směr USB na RS232:, zdroj: usb_function_ftdi.c

4.5.7.2 Handshaking na úrovni ovladače

Handshaking na úrovni ovladače je implementován ze směru USB na port RS232 ve zdrojovém souboru „usb_function_ftdi.c“ funkcí „USBCheckFTDIRequest“ a požadavkem ovladače „SIO_SET_MODEM_CTRL_REQUEST“ na nastavení linek DTR a RTS, který byl blíže popsán v kapitole 3.4.7.2 „*SIO_SET_MODEM_CTRL_REQUEST 0x01h*“ na straně 41. Funkční část kódu je zobrazena ve výpise 12 „*HW handshaking ovladače, směr USB na RS232:, zdroj: usb_function_ftdi.c*“ na straně 72.

Ve směru RS232 na port USB je handshaking implementován uvnitř programové funkce *ProcessIO* v podmínce *IF4:*, kdy se pravidelně emulátor dotazuje na změnu stavu na linkách DSR a CTS porovnáním s předchozím zjištěným stavem. V případě odhalení změny oproti předcházejícímu stavu je tato informace ihned odeslána ovladači na port USB. Na základě této informace se může ovladač následně rozhodnout rozhodnout o pozastavení a následném obnovení posílání dat zpět na převodník zasláním požadavku „SIO_SET_MODEM_CTRL_REQUEST“. Funkční část kódu je zobrazena ve výpise 13 „*HW handshaking ovladače, směr RS232 na USB:, zdroj: main.c*“ na straně 73.

```
// IF4:
if (FlowControlFlag.RTSCTS == 1 || FlowControlFlag.DTRDSR == 1) {
   BYTE ls;
   ls = GetLineStatus();
   if (ls != lastLineStatus) {
      FTDITx();
   }
   lastLineStatus = ls;
}
```

Výpis 13: HW handshaking ovladače, směr RS232 na USB:, zdroj: main.c

4.5.8 Výpočty

Ve firmware byly použity dva výpočty. Jedním je nastavení rychlosti portu RS232 tzv. baudrate interního modulu EUSART mikroprocesoru PIC18F46J50. Druhým výpočtem je nastavení modulu časovače mikroprocesoru použitého u smyčky Latency Timeru, který řídí cyklus automatického odesílání stavu linek portu RS232 a stavu modemu na port USB. Nyní si oba výpočty představíme.

4.5.8.1 Výpočet pro nastavení Latency Timeru

Časovač Latency Timer je použit v programové funkci *ProcessIO* jejíž práce byla popsána v kapitole 4.5.5 „*ProcessIO*“ na straně 66. Latency Timer je implementován pomocí modulu časovače *TIMER0* mikroprocesoru PIC18F46J50. Princip a popis modulu časovače *TIMER0* byl představen v kapitole 2.5.6.1 „*TIMER0*“ na straně 17.

Modul časovače je nutné nejdříve nastavit na požadovanou činnost. Modul časovače je nastaven při inicializaci emulátoru v programové funkci *UserInit*. Modul časovače *TIMER0* může pracovat v režmu časovače nebo čítače. Režim časovače je zvolen bitem *T0CS = 0*. Dále je zvolen 16-ti bitový režim časovače bitem *T08BIT = 0*. Nebudeme používat při přetečení časovače přerušení mikroprocesoru, protože časování smyčky Latency Timeru není kritické, nastavíme tedy *TMR0IF = 0*. Dále budeme potřebovat k práci časovače využít děličky „prescalleru", povolíme tedy prescaller bitem *PSA = 0* a přidělíme mu dělící poměr bitem *T0PS = 0b101* na hodnotu 1:64, která při frekvenci oscilátoru *FOSC = 48MHz* nám umožní dosáhnout požadované časové smyčky Latency Timeru 1 až 255 milisekund, viz dále.

Firmware emulátoru si pamatuje aktuálně nastavenou velikost Latency Timeru v proměnné *ActualLatencyTimer*. Tato proměnná je při startu firmware emulátoru nastavena na výchozích 16ms.

Samotné nastavení registrů mikroprocesoru podle požadované velikosti Latency Timeru v rozmezí 1 až 255 milisekund provádí programová funkce *SetLatencyTimer(WORD number_of_miliseconds)* jejíž bližší práci si nyní popíšeme.

Poznámka 4.12 Důležitým pojmem časovače je jeho „přetečení". V 16-ti bitovém režimu přeteče časovač *TIMER0* při přechodu hodnoty jeho registrů z 0xFFFF do 0x0000. Pokud se tak stane, je nastaven příznakový bit *TMR0IF = 1*. Tento bit sleduje právě podmínka *IF:3* v programové funkci *ProcessIO*. Tato podmínka byla vysvětlena v kapitole 4.5.5 „*ProcessIO*" na straně 66.

Pro správnou práci časovače je tedy nutné nastavit jeho počáteční stav (výchozí hodnoty jeho registrů) tak, aby po tom co začne počítat s hodinovými impulsy mikroprocesoru, dosáhl po svém přetečení předem stanoveného času, v našem případě času Latency Timeru.

Nejprve si tedy spočítáme počet kroků, které musí časovač napočítat, aby uběhl čas 1 milisekundy. Za výchozí hodnoty jsou frekvence mikroprocesoru *FOSC*, hodnota jednoho kroku časovače *FOSC/4* a hodnota prescalleru *PRESC*.

$$FOSC = 48MHz; PRESC = 64$$

Pro 1 milisekundu musí čítač provézt celkem kroků:

$$48000000/4/64/1000 = 187,5$$

Pro 256 milisekund potom musí čítač provézt celkem kroků:

$$48000000/4/64/1000 * 256 = 48000$$

Z výše uvedeného výpočtu je patrno, že počet kroků je v rozsahu dvoubajtového čísla, tedy že jsme správně zvolili hodnotu prescalleru velikosti 64 a že budeme potřebovat použít časovač v 16-ti bitovém režimu, který jsme již výše také správně zvolili.

Nyní ještě zbývá správně nastavit výchozí hodnotu časovače, od které bude počítat, nastavením jeho registrů *TMR0L* a *TMR0H* tak, aby do přetečení uběhl požadovaný čas.

Obecný výpočet je následující:

$$TIMER0 = 65535 - (48000000/4/64 * number\_of\_miliseconds/1000)$$

Dvoubajtová vypočítaná hodnota ve firmware použitá v proměnné *TIMER16B_VAL* se uloží nejdříve do horního bajtu časovače a následně do dolního bajtu časovače. Časovač lze nyní spustit nastavením bitu *TMR0ON = 1*.

Použití výpočtu Latency Timeru je zobrazeno na výpise kódu 14 „*Části kódu pro výpočet hodnoty časovače TIMER0, zdroj: main.c*“ na straně 75. Spuštění a zastavení časovače je zobrazeno na výpise kódu 15 „*Makra pro spuštění a zastavení časovače TIMER0, zdroj: main.c*“ na straně 75.

```
WORD ActualLatencyTimer = 16;
WORD_VAL TIMER16B_VAL = 0;

void UserInit(void) {

    INTCONbits.TMR0IF = 0;
    T0CONbits.T08BIT = 0;
    T0CONbits.T0CS = 0;
    T0CONbits.PSA = 0;
    T0CONbits.T0PS = 0b101;

    SetLatencyTimer((WORD) ActualLatencyTimer);
}

void SetLatencyTimer(WORD number_of_miliseconds) {

    TIMER16B_VAL.Val = 65535 - (48000000 / 4 / 64 * number_of_miliseconds / 1000);
    TMR0H = TIMER16B_VAL.byte.HB;
    TMR0L = TIMER16B_VAL.byte.LB;
    ActualLatencyTimer = number_of_miliseconds;
}
```

Výpis 14: Části kódu pro výpočet hodnoty časovače *TIMER0*, zdroj: main.c

```
#define LATENCY_TIMER_START {TMR0H = TIMER16B_VAL.byte.HB; TMR0L = TIMER16B_VAL.
    byte.LB; INTCONbits.TMR0IF = 0; T0CONbits.TMR0ON = 1;}
#define LATENCY_TIMER_STOP T0CONbits.TMR0ON = 0;
```

Výpis 15: Makra pro spuštění a zastavení časovače *TIMER0*, zdroj: main.c

4.5.8.2 Výpočet pro nastavení baudrate

Pokud USB ovladač FTDI pošle požadavek na nastavení rychlosti portu RS232 tzv. „baudrate“, firmware emulátoru musí na tento požadavek reagovat správným nastavením modulu EUSART mikroprocesoru PIC18F46J50.

Poznámka 4.13 Požadavek na nastavení baudrate modulu EUSART zpracovává programová funkce *USBCheckFTDIRequest* představená v kapitole 4.5.6 „*Zpracování požadavků FTDI ovladače*“ na straně 68. Fyzicky byl modul EUSART pospán v kapitole 2.5.6.2 „*EUSART*“ na straně 18.

Modul EUSART je nutné nejdříve nastavit na požadovanou činnost. Modul je nastaven hned při inicializaci emulátoru v programové funkci *InitializeUSART*. Protože kmitočet našeho mikroprocesoru emulátoru je 48MHz a potřebujeme pokrýt rychlosti baudrate v rozmezí 0.3kbaud až 115.2kbaud, musíme nastavit baudrate generátor mikroprocesoru PIC18F46J50 na 16-ti bitový režim nastavením bitu *BRG16 = 1* v registru *BAUDCON*. Dále je potřeba povolit asynchronní režim v registru *TXSTA* a *RCSTA*.

Nakonec je potřeba nastavit registry *SPBRG* a *SPBRGH* podle požadované rychlosti baudrate.

Baudrate pro nastavený režim modulu EUSART se vypočítá následovně:

$$baudrate = FOSC/[4(n+1)]$$

kde n je požadovaná 16-ti bitová hodnota *[SPBRGH:SPBRG]*. My však požadovaný baudrate známe, upravíme tedy vzorec pro výpočet hodnoty n 16-ti bitového registru *[SPBRGH:SPBRG]*. Vzorec je následující:

$$n = (48000000/4)/baudrate - 1;$$

Za baudrate se dosazují standardní rychlosti sériového portu RS232 podle požadavků USB ovladače a vypočítávají se hodnoty pro naplnění registru *[SPBRGH:SPBRG]*. Ikdyž originální převodník dosahuje rychlosti až 1Mbaud, s emulátorem a mikroprocesorem PIC18F46J50 lze dosáhnout maximální rychlosti RS232 portu 115.2kbaud. Pokud se pokusí USB ovladač nastavit rychlost vyšší, je progamově zajištěno, že rychlost portu RS232 se nezmění a zůstane posledně nastavená.

Část kódu pro výpočet baudrate jak je implementován v emulátoru je ve výpise 16 „*Výpočet baudrate modulu EUSART, zdroj: main.c, usb_function_ftdi.c*“ na straně 76.

```
void InitializeUSART(void) {

    TXSTA = 0x24; // TX enable BRGH=1
    RCSTA = 0x90; // Single Character RX
    SPBRG = 0x71;
    SPBRGH = 0x02; // 0x0271 for 48MHz -> 19200 baud
    BAUDCON = 0x08; // BRG16 = 1
    c = RCREG; // read
}

DWORD_VAL dwBaud;

void USBCheckFTDIRequest() {

     ...
case SIO_SET_BAUDRATE_REQUEST:
     ...
    dwBaud.Val = (DWORD) (GetSystemClock() / 4) / line_coding.dwDTERate.Val - 1;
    SPBRG = dwBaud.v[0];
    SPBRGH = dwBaud.v[1];
     ...
    }
}
```

Výpis 16: Výpočet baudrate modulu EUSART, zdroj: main.c, usb_function_ftdi.c

4.5.9 Výpis funkcí zdrojových souborů

Jako poslední kapitolu části 4.5 *„Realizace firmware emulátoru“* na straně 55 bych zde uvedl seznam programových funkcí emulátoru použitých ve zdrojových souborech „main.c“ a „usb_function_ftdi.c“.

```
void main(void);
void ProcessIO(void);
void SetLatencyTimer(WORD);
static void InitializeSystem(void);
void UserInit(void);
void InitializeUSART(void);
void BlinkUSBStatus(void);
void putcUSART(char c);
unsigned char getcUSART();
void PurgeRXbuffer(void);
void PurgeTXbuffer(void);
```

Výpis 17: Výpis programových funkcí souboru: main.c

```
void USBCheckFTDIRequest(void);
void USBFTDIInitEP(void);
void GetModemLineStatus(WORD_VAL *status);
BYTE GetLineStatus(void);
BYTE getsFTDIUSB(char *buffer, BYTE len);
void putsFTDIUSB(char *buffer, BYTE len);
BOOL USER_USB_CALLBACK_EVENT_HANDLER(USB_EVENT event, void *pdata, WORD size);
```

Výpis 18: Výpis programových funkcí souboru: usb_function_ftdi.c

5 Testování emulátoru

Testování je jednou z disciplín, která je nutnou součástí vývoje jakéhokoliv prototypu nebo jeho části, ať už se jedná o testování hardware nebo software. I my zde podrobíme emulátor některým testům, kterými se pokusíme ověřit správný návrh a výslednou funkčnost emulátoru.

5.1 Hardwarová konfigurace testů

Testy byly prováděny ve dvou hardwarových konfiguracích.

První konfigurací byl hardwarový loopback na portu RS232 emulátoru. Znaky odeslané z portu USB na pin Tx portu RS232 emulátoru se vracely díky vzájemnému propojení pinů Rx a Tx portu RS232 po vlastním pinu Rx zpět do portu USB. Fotografie této konfigurace je na obrázku 26 „*Hardware emulátoru - RS232 port loopback, zdroj: autor*“ na straně 128.

Druhou konfigurací byl hardwarový loopback přes originální FTDI převodník. Znaky odeslané z portu USB 1 emulátoru na pin Tx portu RS232 emulátoru byly přijaty na pinu Rx portu RS232 originálního FTDI převodníku a přeneseny na druhý port USB 2. V opačném směru znaky odeslané z portu USB 2 originálního FTDI převodníku na vlastní pin Tx portu RS232 byly přijaty na opačné straně na pinu Rx portu RS232 emulátoru a přeneseny na port USB 1. Takto vznikla ve druhé kategorii testovací aparatura dvou zařízení propojených ve smyčce emulátor/originální FTDI převodník ([USB1 - RS232] x [RS232 -USB2]). Fotografie této konfigurace je na obrázku 27 „*Hardware emulátoru - loopback přes originální FTDI převodník, zdroj: autor*“ na straně 129.

Poznámka 5.1 Emulátor má port RS232 zapojen přes modul PmodRS232, který je konfigurován jako DCE (slave) zařízení, připojuje se tedy přímým kabelem k master zařízení, například RS232 portu počítače, pokud je jím vybaven nebo k originálnímu FTDI převodníku, který je koncepčně konfigurován jako master zařízení. Zda-li je zařízení master nebo slave jednoduše stanovuje vzájemné přehození linek Rx/Tx, RTS/CTS a DSR/DTR.

5.2 Ověření kompatibility v OS

Emulátor byl testován v následujících operačních systémech:

- Windows 7 Professional 32bit
- Windows XP 32bit
- Linux Ubuntu 11.10 32bit

- Mac OS X Lion 10.7.3 (11D50) 64bit

Hlavním operačním systémem, pod kterým byla prováděna celá škála dále uvedených testů, byl operační systém Windows 7 Professional, který byl zároveň operačním systémem použitým pro vývoj emulátoru. U ostatních operačních systémů byly testy omezeny, protože byl vždy nainstalován originální ovladač FTDI, který zabezpečuje kompatibilitu v různých operačních systémech. Pokud byl emulátor důkladně otestován v jednom operačním sytému, je tedy pravděpodobné, že bude ve stejných testech správně pracovat i v ostatních operačních systémech.

Emulátor byl také otestován v mezipropojení následujících operačních systémů:

- Windows 7 na Mac OS X
- Linux Ubuntu na Mac OS X

Vzhledem k tomu, že mi byly umožněny testy pod operačním systémem Mac OS X, zahrnul jsem do testů propojení tohoto operačního systému se systémem Linux, který jsem bootoval ve verzi Live na mém notebooku. Dále jsem otestoval propojení mezi operačním systémem Mac OS X a operačním systémem Windows 7, který byl nativním systémem mého notebooku. Mezipropojení obou operačních systémů proběhlo přes druhou konfiguraci „emulátor - originální FTDI převodník“.

5.3 Testování spolehlivosti emulátoru

Do testů spolehlivosti byly zahrnuty následující činnosti a testy emulátoru:

- Připojení a odpojení emulátoru
- Plynulý přísun znaků

Připojení a odpojení emulátoru bylo realizováno opakovaným připojením a odpojením emulátoru od sběrnice USB. Tento krok byl prováděn opakovaně 10x za sebou. Po každém připojení emulátoru bylo ve *Správci zařízení* v operačním systému zkontrolováno, zda se emulátor úspěšně připojil a získal virtuální COM port. Po odpojení emulátoru bylo ve *Správci zařízení* v operačním systému zkontrolováno odebrání zařízení a komunikačního COM portu. Pokud byly všechny za sebou provedené testy připojení úspěšné, dostal tento dílčí test status „pass“.

Testování plynulého přísunu znaků se týká přenášení znaků mezi porty USB a RS232 emulátoru. Účelem tohoto testu bylo získat přehled, zda-li znaky přecházejí po cestě z

portu USB do portu RS232 a naopak z portu RS232 do portu USB plynule, bez viditelného zpomalování či jejich zasekávání. Součástí testu byla i nahodilá ztráta funkčnosti celého emulátoru, jejímž výsledkem by mohlo být přerušení toku znaků. Tento test byl prováděn při všech testovaných rychlostech portu RS232. Pokud byl jednotlivý test úspěšný, dostal status „pass“.

5.4 Testování stability emulátoru

Do testů stability byly zahrnuty následující činnosti a testy emulátoru:

- Přepínání baudrate
- Re-enumerace

Emulátor byl testován při různých rychlostech portu RS232. Testem stability bylo prověřováno, že přepnutím na jinou rychlost portu RS232 nedošlo ke ztrátě funkčnosti emulátoru. Dále bylo zjišťováno, zda-li nedochází k samovolné re-enumeraci zařízení emulátoru, tedy jeho nahodilému odpojení od operačního systému a případné jeho opětovné připojení a to v průběhu celé práce emulátoru a prováděných testů. Úspěšnost testů byla zaznamenána výsledkem „pass“.

5.5 Testování propustnosti emulátoru

Do testů propustnosti byly zahrnuty následující činnosti a testy emulátoru:

- Baudrate rychlost (300baud až 115.2kbaud)
- Handshaking

Jednotlivé testy přesunu znaků emulátoru mezi porty USB a RS232 byly prováděny vždy pro každou rychlost portu RS232. Testu se účastnily rychlosti 300baud až 115.2kbaud. Rychlost 115.2kbaud je nejvyšší možná rychlost, které můžno tento emulátor dosáhnout. Tato rychlost je omezena rychlostí samotného procesoru PIC18F46J50 a jeho modulu EUSART. Úspěšnost každého dílčího testu byla hodnocena statusem „pass“.

Každý výše uvedený test byl proveden pro nastavení handshakingu na *none*, *software (XON/XOFF)* a *hardware (RTS/CTS)*. Úspěšnost testu byla hodnocena statusem „pass“.

Poznámka 5.2 I když firmware emulátoru obsahuje hardwarový handshaking na linkách DTR a DSR, tak vzhledem k hardwarové konfiguraci emulátoru a s přihlédnutím,

že DTR/DSR handshaking není handshaking podporovaný standardem RS232 a ani ovladače v operačním systému Windows, Linux, či Mac OS X nenabízí možnost hardwarového handshakingu na linkách DTR/DSR, nebyl tento typ handshakingu v emulátoru testován.

5.6 Způsob provádění testů

Testy byly prováděny odesíláním znaků z programu terminálu a jejich zpětným přijetím opět v programu terminálu. Byly testovány celkem čtyři způsoby odesílání:

1. Prosté manuální psaní textu v terminálu
2. Posílání krátkého textu ze souboru „rs232_data_text_kratky.txt“
3. Posílání dlouhého textu ze souboru „rs232_data_text_dlouhy.txt“
4. Posílání velmi dlouhého textu ze souboru „rs232_data_text_velmi.txt“

Základním testem funkčnosti emulátoru bylo posílání jednotlivých znaků z manuálního psaní v aplikaci terminálu, kdy psaním znaků docházelo k jejich odesílání. Znaky byly přijímány zpět do okna terminálu pokud se test týkal první testovací konfigurace hardwarového loopbacku na portu RS232 nebo byly přijímány do okna druhého terminálu v případě druhé konfigurace, kdy byly vzájemně propojeny emulátor a originální FTDI převodník.

Následujícím testem bylo odesílání textových dat pomocí funkce „Oddeslat textový soubor... “ aplikace terminálu, která automaticky odešle vybraný soubor na port RS232. Tento test byl rozdělen celkem do tří délek souborů. Účelem bylo otestovat, jakým způsobem se emulátor chová při rychlém přísunu dat potřebných k odeslání a to malé délky, větší délky a dlouhého souboru s obsahem odesílaných dat.

V průběhu výše uvedených testů se prověřovalo, zda-li nedochází ke ztrátě odeslaných dat jejich porovnáním v okně terminálu, který data zpět přijímal. Pokud byl jednotlivý dílčí test úspěšný, byl mu přidělen status „pass“.

Poznámka 5.3 Délka krátkého textu byla 18 znaků. Délka dlouhého textu byla 187 znaků. Velmi dlouhý text vznikl opakovaným nakopírováním dlouhého textu, celkem měl velmi dlouhý text délku 2067 znaků.

5.7 Výsledky testování

V této kapitole jsou uvedeny výsledky testování emulátoru. Protože testování probíhalo v průběhu celého vývoje emulátoru, byly reálně obsáhlé a důkladné a nakonec také úspěšné.

Všechny testy provedené pod operačním systémem Windows 7, který byl vývojovou platformou, jsou shrnuty a zaznamenány do výsledných tabulek. Výsledky testů ostatních operačních systémů jsou zmíněny dále v textu, ale vzhledem k obsáhlosti a podobnosti s testy pod operačním systémem Windows 7 již nejsou uváděny v tabulkách.

5.7.1 Výsledky - test spolehlivosti

Při testu připojení a odpojení emulátoru bylo provedeno celkem deset opakování za sebou. Při každém pokusu o připojení k portu USB bylo zařízení emulátoru správně enumerováno a rozpoznáno operačním systémem, což bylo zkontrolováno ve správci zařízení. Při odpojení emulátoru byly ovladače ze správce zařízení operačním systémem odstraněny. Všechny opakované pokusy byly úspěšné, test tedy plně vyhověl a prošel se statusem „pass“.

Výsledky testů spolehlivosti plynulého přísunu znaků byly logicky zahrnuty do kapitoly 5.7.2 „*Výsledky - RS232 loopback*“ na straně 83 a kapitoly 5.7.3 „*Výsledky - loopback emulátor a originální převodník FTDI*“ na straně 83.

5.7.2 Výsledky - RS232 loopback

Všechny testy konfigurace, kdy emulátor pracuje přes hardwarový loopback na portu RS232, ať již s handshakingem nebo bez něj, proběhly úspešně. Výsledek testů je zaznamenán v příloze v tabulce 15 „*Test emulátoru - konfigurace RS232 port loopback, zdroj: autor*“ na straně 114.

5.7.3 Výsledky - loopback emulátor a originální převodník FTDI

Všechny testy konfigurace, kdy je emulátor propojen z jednoho USB portu přes originální FTDI převodník do druhého USB portu, ať již s handshakingem nebo bez něj, proběhly úspešně. Výsledek testů je zaznamenán v příloze v tabulce 16 „*Test emulátoru - konfigurace loopback přes originální FTDI převodník, zdroj: autor*“ na straně 115.

5.7.4 Výsledky - ostatní operační systémy

Emulátor byl testován také pod operačními systémy Windows XP, Linux Ubuntu a Mac OS X. Testy byly provedeny způsobem manuálního psaní textu v terminálu. Při testu přes hardwarový loopback na portu RS232 byl použit jeden terminál. Při testu přes originální FTDI převodník bylo psaní textu prováděno mezi dvěma terminály. Testování proběhlo na náhodně vybraných rychlostech a na všech druzích handshakingu. Vždy však byla k

testování minimálně vybrána nejnižší rychlost 300baud a nejvyšší rychlost 115.2kbaud. Všechny testy proběhly úspěšně „pass“, zaznamenání do tabulky nebylo provedeno.

5.7.5 Výsledky - mezipropojení operačních systémů

Emulátor byl testován také v konfiguraci mezipropojení operačních systémů mezi dvěma notebooky za použití propojení zařízení emulátoru a originálního FTDI převodníku. Všechny testy proběhly úspěšně „pass“, zaznamenání do tabulky nebylo provedeno. Obsah oken programů terminálů testovaného propojení je zobrazen v příloze na obrázku 25 „*Obsah terminálů mezipropojení OS Mac OS X a Linux Ubuntu, zdroj: autor*“ na straně 126.

Poznámka 5.4 Testy pod operačním systémem MaC OS X proběhly na notebooku MacBook Pro Mid 2010, konfigurace Intel Core 2 Duo, 8GB RAM.

Poznámka 5.5 Při propojení operačního systému Mac OS X s operačním systémem Windows 7 byl emulátor zasunut v USB portu notebooku pracujícího pod operačním systémem MaC OS X. Při propojení operačního systému Mac OS X s operačním systémem Linux Ubuntu byl emulátor zasunut v USB portu notebooku pracujícího pod operačním systémem Linux Ubuntu.

5.8 Závěr testování emulátoru

Všechny provedené testy byly úspěšné. Lze prohlásit, že úkol „Programová implementace USB-RS232 převodníku v procesoru Microchip“ byl vyřešen.

5.9 Omezení emulátoru

Emulátor převodníku FTDI má i svá omezení a některá z nich jsem již uváděl v předchozích kapitolách. Nyní je zde pro úplnost shrnuji.

Emulátor podporuje pouze konfiguraci RS232 portu 8N1 (počet bitů = 8, parita = žádná, počet stopbitů = 1). Důvodem je, že modul EUSART mikroprocesoru PIC18F46J50 v asynchronním režimu podporuje standardní formát (1 start bit, data 8 nebo 9 bitů, 1 stop bit) a hardwarově nepodporuje paritu. Naopak originální FTDI převodník podporuje délku dat 5 až 8 bitů, nikoliv však 9 a všechny typy parity plus stop bity 1, 1.5 a 2.

Originální převodník FTDI podporuje rychlosti portu RS232 až do 921kbaud. S emulátorem a použitým mikroprocesorem PIC18F46J50 a jeho modulem EUASRT lze však technicky dosáhnout maximální rychlosti pouze 115.2kbaud.

Emulátor má implementovanou interní paměť EEPROM jako originální převodník FTDI. Tato paměť je však vytvořena prostou kopií originálního obsahu paměti převodníku v paměti ROM emulátoru. Emulátor tedy nepodporuje funkce zápisu do paměti EEPROM jako originální FTDI převodník.

5.10 Licence ovladačů FTDI

Ovladače pro originální FTDI převodník dodává výrobce do různých operačních systémů. V případě ovladačů pro operační systém Windows, uvádí výrobce následující licenční omezení, které je součástí instalace souboru „ftdibus.inf“ :

- FTDI DRIVERS MAY BE USED ONLY IN CONJUNCTION WITH PRODUCTS BASED ON FTDI PARTS.

Protože emulátor nekopíruje hardwarovou část originálního převodníku, ale pouze softwarově emuluje některé jeho funkce a některé dokonce omezeně viz. kapitola 5.9 „*Omezení emulátoru*“ na straně 84 a s přihlédnutím k faktu, že byl emulátor vytvořen za účelem studijní práce nikoliv komerčního využití, mám za to, že licenční podmínky nejsou v tomto případě porušovány.

V operačním systému Linux a za použití open source ovladačů, například knihovny LibFTDI viz. reference [21], je použití emulátoru taktéž naprosto legální.

6 USB implementace dvojitý převodník

Součástí zadání této práce je také „zvážení možnosti implementace dvou a více převodníků jedním mikropočítačem“.

V průběhu práce na vývoji emulátoru jsem zjistil, že možnost implementace více převodníků jedním mikroprocesorem je možná ve spolupráci s FTDI ovladači, které jsou na práci víceportového převodníku uzpůsobeny, protože výrobce FTDI má ve své nabídce právě obvody podporující víceportová zařízení. Jedním z takových obvodů je například obvod FT2232D nebo obvod FT4232H, kde první číslo v názvu za „FT“ značí počet portů. V kapitole 3.4.7 „*Popis USB protokolu FTDI - Control Requests*“ na straně 40, kde jsme si popisovali požadavky ovladače, jsme také ke konci jednotlivých požadavků zmiňovali, že „hodnota *wIndex* určuje RS232 port, kterého se požadavek týká“.

Samozřejmě pokud umíme otevřít více současných cest ze směru USB do zařízení emulátoru, nemusíme nutně všechny cesty využít pro následující převod dat na port RS232. Otevřený virtuální komunikační COM port na straně počítače lze využít i k posílání dat do emulátoru a zpět a nemusí být nutně průchozí dále na druhý port RS232. Data lze využít například pro jinou činnost emulátoru anebo přeposílat na jiný typ portu například I^2C. Protože tedy není nutno každou další cestu využít pro RS232 port, budeme dále označovat jednu cestu jako „kanál“ a odkazovat se na emulátor jako „vícekanálový“.

Níže bude popsán způsob implementace více komunikačních kanálů do našeho emulátoru. Implementace bude ověřena úpravou stávajícího firmware na dvoukanálový emulátor. Jeden kanál bude sloužit jako průchozí převodník USB na port RS232 jako je u první verze emulátoru. Aby nemusela být prováděna hardwarová úprava emulátoru, druhý kanál emulátoru bude pracovat jako softwarový loopback, kdy znak přijatý z tohoto kanálu bude vrácen na tento kanál zpět. Demonstrace softwarového loopbacku na druhém kanálu bude plně dostačující pro prověření imlementace více převodníků jedním emulátorem. Pokud data dorazí na druhém kanálu do emulátoru, lze s nimi ve firmware aplikovat jakoukoliv požadovanou funkci. Schematické zobrazení realizace kanálů tímto emulátorem je vyobrazena v příloze na obrázku 24 „*Realizace kanálů dvojího převodníku, zdroj: autor*“ na straně 123.

Poznámka 6.1 FTDI obvod FT2232D je třetí generací řady FT2232 a nástupcem svých předchůdců FT2232C a FT2232L. V následujícím textu se tedy budeme odkazovat na poslední typ FT2232D. Z pozice ovladačů se nic nemění, vždy je zaváděn společný ovladač pro obvod typu FT2232 a konkrétní typ (D/C/L) se na úrovni ovladače nerozlišuje.

6.1 Úprava deskriptoru emulátoru

Aby mohl být operačním systémem rozpoznán vícekanálový emulátor a zaveden pro něj odpovídající ovladač, musí být po připojení do USB portu správně enumerován. Protože enumeraci zajišťuje deskriptor zařízení, musí být správně nakonfigurován.

Deskriptor pro vícekanálový emulátor vytvoříme úpravou deskriptoru emulátoru FTDI převodníku tak, aby odpovídal originálnímu dvouportovému FTDI převodníku FT2232D. Změníme Product ID „PID = 0x6010“, Device release number „bcdDevice = 0x0500“ , přidáme jeden interfejs se dvěma novými endpointy a upravíme délku konfiguračního deskriptoru na 55 bajtů. Vyjmenovaná úprava je zobrazena ve výpise kódu 19 „*Úprava deskriptoru pro duální převodník, zdroj: autor*“ na straně 88.

```
/* Device Descriptor */
ROM USB_DEVICE_DESCRIPTOR device_dsc=
{
    0x12,                   // sizeof(USB_DEVICE_DESCRIPTOR)
    USB_DESCRIPTOR_DEVICE, // DEVICE descriptor type
    0x0200,                 // USB Spec Release Number in BCD format
    0x00,                   // Class Code
    0x00,                   // Subclass code
    0x00,                   // Protocol code
    USB_EP0_BUFF_SIZE,  // Max packet size for EP0, see usb_config.h
    0x0403,                 // Vendor ID
    0x6010,                 // Product ID
    0x0500,                 // Device release number in BCD format / FT2232D
    0x01,                   // Manufacturer string index
    0x02,                   // Product string index
    0x03,                   // Device serial number string index
    0x01                    // Number of possible configurations
};

/* Configuration 1 Descriptor */
ROM BYTE configDescriptor1[]={
    /* Configuration Descriptor */
    0x09,                   //sizeof(USB_CFG_DSC)
    USB_DESCRIPTOR_CONFIGURATION, // CONFIGURATION descriptor type
    55,0,                   // Total length of data for this cfg
    2,                      // Number of interfaces in this cfg
    1,                      // Index value of this configuration
    0,                      // Configuration string index
    _DEFAULT,               // Bus Powered Attributes, see usb_device.h
    50,                     // Max power consumption (x2 mA)

    /* Interface Descriptor */
    0x09,                   //sizeof(USB_INTF_DSC),
    USB_DESCRIPTOR_INTERFACE, // INTERFACE descriptor type
    0,                      // Interface Number
    0,                      // Alternate Setting Number
```

```
    2,                      // Number of endpoints in this intf
    0xFF,                   // Class code
    0xFF,                   // Subclass code
    0xFF,                   // Protocol code
    2,                      // Interface string index

    /* Endpoint Descriptor */
    0x07,                   //sizeof(USB_EP_DSC)
    USB_DESCRIPTOR_ENDPOINT,//Endpoint Descriptor
    _EP01_IN,               //EndpointAddress
    _BULK,                  // Attributes
    0x40,0x00,              //size
    0x00,                   // Interval

    /* Endpoint Descriptor */
    0x07,                   //sizeof(USB_EP_DSC)
    USB_DESCRIPTOR_ENDPOINT,//Endpoint Descriptor
    _EP02_OUT,              //EndpointAddress
    _BULK,                  // Attributes
    0x40,0x00,              //size
    0x00,                    // Interval

    /* Interface Descriptor */
    0x09,                   //sizeof(USB_INTF_DSC),
    USB_DESCRIPTOR_INTERFACE, // INTERFACE descriptor type
    1,                      // Interface Number
    0,                      // Alternate Setting Number
    2,                      // Number of endpoints in this intf
    0xFF,                   // Class code
    0xFF,                   // Subclass code
    0xFF,                   // Protocol code
    2,                      // Interface string index

    /* Endpoint Descriptor */
    0x07,                   //sizeof(USB_EP_DSC)
    USB_DESCRIPTOR_ENDPOINT,//Endpoint Descriptor
    _EP03_IN,               //EndpointAddress
    _BULK,                  // Attributes
    0x40,0x00,              //size
    0x00,                    // Interval

    /* Endpoint Descriptor */
    0x07,                   //sizeof(USB_EP_DSC)
    USB_DESCRIPTOR_ENDPOINT,//Endpoint Descriptor
    _EP04_OUT,              //EndpointAddress
    _BULK,                  // Attributes
    0x40,0x00,              //size
    0x00                    // Interval
};
```

Výpis 19: Úprava deskriptoru pro duální převodník, zdroj: autor

Poznámka 6.2 Protože jsem měl původní jednokanálový emulátor v operačním systému Windows 7 registrován pod sériovým číslem FTR69S36, po úpravě deskriptoru emulátoru nebyl vícekanálový emulátor správně identifikován a nebyly pro něj nalezeny v systému ovladače ani po jejich odinstalování a novém nainstalování. Až po odebrání ovladače utilitou *USBDeview v 1.97 (www.nirsoft.net)* ve které lze odebrat ovladač podle sériového čísla a po novém připojení vícekanálového emulátoru k USB portu, byly ovladače správně nainstalovány a zařízení enumerováno na dva virtuální COM porty.

6.2 Úprava konfiguračních souborů

Dále je třeba v původním emulátoru upravit konfigurační soubor *usb_config.h*. Je nutno zvětšit počet endpointů a počet konfiguračních deskriptorů, konfigurovat nové endpointy a přidat makra pro jejich povolení ve firmware. Vyjmenovaná úprava je zobrazena ve výpise kódu 20 „*Úprava konfiguračního souboru pro duální převodník, zdroj: autor*“ na straně 90.

```
#define USB_MAX_EP_NUMBER  4
#define USB_MAX_NUM_INT  2 // For tracking  Alternate  Setting
#define FTDI_DATA_EP_IN2        0x03 //_EP03_IN|USB_IN_EP
#define FTDI_DATA_IN_EP_SIZE  0x40
#define FTDI_DATA_EP_OUT2  0x04 //_EP04_OUT|USB_OUT_EP
#define FTDI_DATA_OUT_EP_SIZE 0x40
#define ENABLE_FTDI_DATA_EP_IN2() USBEnableEndpoint(FTDI_DATA_EP_IN2,
    USB_IN_ENABLED | USB_HANDSHAKE_ENABLED | USB_DISALLOW_SETUP);
#define ENABLE_FTDI_DATA_EP_OUT2() USBEnableEndpoint(FTDI_DATA_EP_OUT2,
    USB_OUT_ENABLED | USB_HANDSHAKE_ENABLED | USB_DISALLOW_SETUP);
```

Výpis 20: Úprava konfiguračního souboru pro duální převodník, zdroj: autor

6.3 Úprava zdrojových souborů

Do zdrojového souboru *main.c* byly přidány bufery pro příjem a odeslání znaku na USB port, přepracováno odesílání dat na USB port v programové funkci *FTDITx2* a do programové funkce *ProcessIO* byla přidána nová programová funkce *ProcessSWLoopback* zabezpečující softwarový loopback, kdy data přijatá z druhého kanálu portu USB jsou odesílána po druhém kanálu v nezměněné podobě zpět na port USB. Obsah programové funkce *ProcessSWLoopback* je zobrazen na výpise kódu 21 „*Implementace sw loopbacku na druhém kanálu, zdroj: autor*“ na straně 91.

```
void ProcessSWLoopback(void) {

    port2receivedLen = getsFTDIUSB2(USB_Out2_Data, sizeof (USB_Out2_Data));
    if (port2receivedLen > 0) { //odešli ihned včetně statusu
        putsFTDIUSB2(USB_Out2_Data, 2 + port2receivedLen);
        port2receivedLen = 0;
    }

}
```

Výpis 21: Implementace sw loopbacku na druhém kanálu, zdroj: autor

Ve zdrojovém souboru *usb_function_ftdi.c* byla přepracována programová funkce *USBFTDIInitEP* pro inicializaci nových endpointů a jejich handle a byly přidány dvě nové programové funkce *getsFTDIUSB2* a *putsFTDIUSB2* pro činnost posílání dat na USB port druhého kanálu. Dále byla přepracována programová funkce *USBCheckFTDIRequest* tak, aby mohla zpracovávat požadavky z různých kanálů podle parametru *wIndex*, který určuje, kterého kanálu se požadavek týká.

6.4 Paměť EEPROM dvoukanálového převodníku

Protože emulovaný dvoukanálový převodník se díky nastavenému deskriptoru přihlásí operačnímu systému jako FTDI obvod FT2232D, předpokládá ovladač přítomnost paměti EEPROM obsahující konfiguraci tohoto obvodu, deskriptor a sériové číslo. Protože jsem však nenalezl originální obsah paměti obvodu FT2232D, vyřadil jsem ji v emulátoru z činnosti. Naštěstí jsou FTDI ovladače nastaveny tak, že lze použít obvod FT2232D a tedy i náš dvoukanálový převodník i bez paměti. Ovladač potom předpokládá, že je obvod FT2232D konfigurován jako dvouportový USB-RS232 sériový převodník.

6.5 Testování dvoukanálového převodníku

Test dvoukanálového převodníku proběhl za současného spuštění dvou terminálových programů. Jeden terminálový program obsluhoval standardní převodník USB na port RS232 na prvním kanálu a druhý terminálový program obsluhoval softwarový loopback na druhém kanálu. Kanály byly střídavě testovány střídáním obsluhy obou terminálů. Obsahy testů byly obdobné jako při testování jednoportového emulátoru uvedené v kapitole 5 „*Testování emulátoru*“ na straně 79.

6.6 Závěr dvojitý převodník

Všechny provedené testy na dvoukanálovém emulátoru byly úspěšné. Lze prohlásit, že úkol „Zvažte možnost implementace dvou a více převodníků jedním mikropočítačem“ v části dvou převodníků byl vyřešen.

7 Více převodníků

Možná dvoukanálový emulátor bude pro někoho přeci jen málo. Jak tedy přidat více převodníků/kanálů do jednoho emulátoru? Zkusme se teoreticky zamyslet nad následujícími úvahami.

7.1 Emulace víceportového FTDI obvodu

První myšlenkou, jak získat víceportový převodník, je emulovat FTDI obvod, který již hardwarově více portů nabízí. Je tedy třeba si vybrat emulaci FTDI obvodu nabízejícího více portů a upravit odpovídajícím způsobem deskriptor emulátoru, podle kterého dochází k enumeraci zařízení v operačním systému a následně příslušně upravit firmware.

Na základě této myšlenky byl vytvořen dvojí převodník představený v kapitole 6 „*USB implementace dvojitý převodník*“ na straně 87. V současnosti lze vytvořit maximálně 4 kanálový převodník emulováním čtyřportového obvodu FT4232H, protože víceportový sériový převodník výrobce FTDI v současné době nenabízí.

7.2 Přidání portů do ovladače

Pokusme se nyní podívat na protější stranu hardware, na ovladač. Zkusme zjistit, jakým způsobem FTDI ovladač pracuje s více porty a víceportovými FTDI převodníky.

7.2.1 Test přidání portů

Jako výchozí emulátor vezmeme náš jednokanálový emulátor a provedeme následující úpravu:

1. Upravíme soubor ovladače
2. Upravíme firmware
3. Zavedeme upravený ovladač

Ovladači je třeba sdělit, že má použít více portů. To provedeme úpravou souboru *ftdibus.inf*. Pro náš jednoportový převodník, který má *PID = 6001*, upravíme soubor ovladače *ftdibus.inf* tak, že za PID identifikátor přidáme nový text &*MI_XX*. Číslo *XX* značí pořadové číslo portů s číslováním od nuly. Výsledná úprava je zobrazena na výpise kódu 22 „*Úprava souboru ftdibus.inf, pokus první, zdroj: autor*“ na straně 94.

```
[FtdiHw]
%USB\VID_0403&PID_6001&MI_00.DeviceDesc%=FtdiBus.NT,USB\VID_0403&PID_6001&MI_00
%USB\VID_0403&PID_6001&MI_01.DeviceDesc%=FtdiBus.NT,USB\VID_0403&PID_6001&MI_01
[FtdiHw.NTamd64]
%USB\VID_0403&PID_6001&MI_00.DeviceDesc%=FtdiBus.NTamd64,USB\VID_0403&PID_6001&
    MI_00
%USB\VID_0403&PID_6001&MI_01.DeviceDesc%=FtdiBus.NTamd64,USB\VID_0403&PID_6001&
    MI_01
[Strings]
USB\VID_0403&PID_6001&MI_00.DeviceDesc="USB Serial Converter A"
USB\VID_0403&PID_6001&MI_01.DeviceDesc="USB Serial Converter B"
```

Výpis 22: Úprava souboru *ftdibus.inf*, pokus první, zdroj: autor

Ve firmware emulátoru provedeme úpravu deskriptoru přidáním nového interfejsu se dvěma endpointy pro druhý kanál a dále tyto endpointy ve firmware povolíme.

Nyní odinstalujeme původní ovladač jeho odebráním ze *správce zařízení* a po opětovném připojení zařízení k USB portu zavedeme nově upravený ovladač. Operační systém však enumeruje pouze jeden port, který je ve *Správci zařízení* pod záložkou *zařízení USB* ohlášen jako „USB Serial Converter“.

Zjištění 7.1 Na vině je USB parametr deskriptoru zařízení *bcdDevice*, podle kterého FTDI ovladač rozeznává typ připojeného zařízení, konkrétně typ použitého obvodu převodníku. Náš emulátor, který emuluje obvod FT232R má parametr „bcdDevice = 0x600“, který značí typ obvodu „R“. Díky tomu ovladač nepočítá s více porty FTDI obvodu, protože obvod typu „R - FT232R“ prostě technicky žádné další hardwarové porty nemá. A nebude je mít, ani když mu to explicitně sdělíme v souboru ovladače *ftdibus.inf* přidáním indexů &*MI_XX*.

Nicméně s našim emulátorem můžeme nastavit parametr „bcdDevice“ podle potřeby a trochu tak zkusit ovladač „ošidit“ malým trikem. Řekneme tedy našemu emulátoru „buď obvodem FT2232D!“ tím, že nastavíme parametr deskriptou zařízení „bcdDevice“ na hodnotu „bcdDevice = 0x500“. To je pro ovladač zpráva, že má co do činění s obvodem FTDI typu FT2232D.

Připojíme tedy zařízení emulátoru znovu do portu USB. Nyní je jíž enumerace zařízení úspěšná a emulátor se ohlásí ve správci USB zařízení jako dva převodníky pro dva porty. Pokud tedy chceme v ovladači manipulovat s více porty, musí si být ovladač jist, že má naproti rovnocenného partnera, FTDI zařízení, které více portů hardwarově podporuje.

7.2.2 Přídání 6-ti portů

V kapitole 7.2.1 „*Test přidání portů*“ na straně 93 jsme zjistili, že nestačí jen přidat porty do ovladače, ale je také nutno zařízení nastavit pomocí parametru deskriptoru „bcdDevice“ na typ FTDI obvodu, který má více než jeden hardwarový port. Dále jsme zjistili, že jsme omezeni typem obvodu FTDI na maximálně 4 porty.

Nicméně jsem se rozhodl otestovat přidání celkem 6-ti portů. Podle kapitoly 7.2.1 „*Test přidání portů*“ na straně 93 nám nyní emulátor pracuje (přesněji se v operačním systému enumeruje) na dva porty podle specifikace obvodu FT2232D, který je hardwarově dvouportový a který naše zařízení emuluje díky triku v nastavení deskriptoru. Z uvedeného vyplývá, že teoreticky nemá smysl další port do ovladače pro obvod FT2232D přidávat, protože jej hardware stejně nepodporuje. Můžeme to zkusit prakticky ověřit našim emulátorem.

Vraťme se tedy zpět k našemu souboru *ftdibus.inf*. Dopišme tedy v souboru ovladače k našemu zařízení *PID = 6001* další parametry &*MI_XX*, celkem 6 portů označených *MI_00* až *MI_05* a odlišme původně výrobcem nastavený text, který se objevuje ve *Správci zařízení* pro *zařízení USB*, abychom měli přehled o zařízení, které jsme si sami definovali. Nastavme text našeho převodníku na „USB Serial Converter AA“ až „USB Serial Converter FF“.

```
[FtdiHw]
%USB\VID_0403&PID_6001&MI_00.DeviceDesc%=FtdiBus.NT,USB\VID_0403&PID_6001&MI_00
%USB\VID_0403&PID_6001&MI_01.DeviceDesc%=FtdiBus.NT,USB\VID_0403&PID_6001&MI_01
%USB\VID_0403&PID_6001&MI_02.DeviceDesc%=FtdiBus.NT,USB\VID_0403&PID_6001&MI_02
%USB\VID_0403&PID_6001&MI_03.DeviceDesc%=FtdiBus.NT,USB\VID_0403&PID_6001&MI_03
%USB\VID_0403&PID_6001&MI_04.DeviceDesc%=FtdiBus.NT,USB\VID_0403&PID_6001&MI_04
%USB\VID_0403&PID_6001&MI_05.DeviceDesc%=FtdiBus.NT,USB\VID_0403&PID_6001&MI_05
[FtdiHw.NTamd64]
%USB\VID_0403&PID_6001&MI_00.DeviceDesc%=FtdiBus.NTamd64,USB\VID_0403&PID_6001&
    MI_00
%USB\VID_0403&PID_6001&MI_01.DeviceDesc%=FtdiBus.NTamd64,USB\VID_0403&PID_6001&
    MI_01
%USB\VID_0403&PID_6001&MI_02.DeviceDesc%=FtdiBus.NTamd64,USB\VID_0403&PID_6001&
    MI_02
%USB\VID_0403&PID_6001&MI_03.DeviceDesc%=FtdiBus.NTamd64,USB\VID_0403&PID_6001&
    MI_03
%USB\VID_0403&PID_6001&MI_04.DeviceDesc%=FtdiBus.NTamd64,USB\VID_0403&PID_6001&
    MI_04
%USB\VID_0403&PID_6001&MI_05.DeviceDesc%=FtdiBus.NTamd64,USB\VID_0403&PID_6001&
    MI_05
USB\VID_0403&PID_6001&MI_00.DeviceDesc="USB Serial Converter AA"
USB\VID_0403&PID_6001&MI_01.DeviceDesc="USB Serial Converter BB"
USB\VID_0403&PID_6001&MI_02.DeviceDesc="USB Serial Converter CC"
USB\VID_0403&PID_6001&MI_03.DeviceDesc="USB Serial Converter DD"
USB\VID_0403&PID_6001&MI_04.DeviceDesc="USB Serial Converter EE"
USB\VID_0403&PID_6001&MI_05.DeviceDesc="USB Serial Converter FF"
```

Výpis 23: Úprava souboru *ftdibus.inf*, pokus druhý, zdroj: autor

Námi dokončená úprava souboru *ftdibus.inf* by měla vypadat jako na výpise kódu 23 „*Úprava souboru ftdibus.inf, pokus druhý, zdroj: autor*“ na straně 95.

Máme upraveny ovladače pro 6 portů, dále tedy postupujeme obdobně jako v kapitole 7.2.1 „*Test přidání portů*“ na straně 93:

1. Upravíme firmware
2. Zavedeme upravený ovladač

Opět upravíme firmware a přidáme nové interfejsy s endpointama do deskriptoru a povolíme endpointy ve firmware.

Nyní odinstalujeme původní ovladač a připojíme zařízení k USB portu. Ve *Správci zařízení* v záložce *Další zařízení* se objevilo celkem 6 zařízení pojmenovaných *USB <-> Serial Cable*, přesně podle obsahu řetězce jako je nastaveno v deskriptoru zařízení emulátoru. Aktualizujeme na tomto zařízení ovladače z adresáře, ve kterém jsme prováděli úpravu souboru *ftdibus.inf*. Díky úpravám, které jsme provedli v inf souboru, se stal ovladač nepodepsaným. To nám nevadí, protože jsme si vědomi provedených úprav a povolíme nainstalovat nepodepsaný ovladač. Tuto akci opakujeme pro dalších 5 *USB <-> Serial Cable* zbývajících zařízení. Po tomto kroku se ve *Správci zařízení* objeví známý soubor řetězců „USB Serial Converter AA“ až „USB Serial Converter FF“, které jsme si sami definovali v souboru *ftdibus.inf*.

Nyní u nově vytvořených položek „USB Serial Converter AA“ až „USB Serial Converter FF“ ve *Vlastnostech* na záložce *Advanced* zaškrtneme volbu *Load VCP*. Tímto sdělíme FTDI ovladači, že má přistupovat k zařízení „USB Serial Converter XX“, (kde XX je pořadové označení AA až FF), jako k virtuálnímu portu. Vlastně nařídíme ovladači zavedení viruálního COM portu pro sériový převodník.

Poznámka 7.1 Původně nebylo možno provézt na druhém až pátém USB zařízení zaškrtnutí volby *Load VCP* z důvodu, že okno pro zaškrtnutí tohoto políčka nebylo možno zpřístupnit a okno *správce zařízení* se zablokovalo a dostalo status operačního systému „Neodpovídá“. Po shlédnutí situace přes program USBTrace byla nalezena odlišnost. Po pravém kliku na zařízení „USB Serial Converter AA“ se ovladač dotázal zařízení emulátoru požadavkem *0x05h* na stav modemu. Když nedostal ovladač ze zařízení emulátoru odezvu, nezpracoval požadavek na otevření okna pro nastavení parametrů a zablokoval správce zařízení. Situace byla napravena úpravou firmware tak, aby správně zpracoval požadavky všech 6-ti portů odesláním stavu modemu na všechny příslušné IN endpointy. Tato úprava firmware byla implementována do programové funkce *putsFTDIUSB*, která je volána z makra *FTDITx()*. Po této úpravě bylo možno povolit virtuální porty u všech zařízení „USB Serial Converter AA“ až „USB Serial Converter FF“.

Po povolení virtuálních portů se ve *Správci zařízení* v záložce *Další zařízení* objeví zařízení pojmenované *USB Serial Port*. Aktualizujeme tedy pro každé toto zařízení ovladače z adresáře, ve kterém jsme prováděli úpravu souboru *ftdibus.inf*.

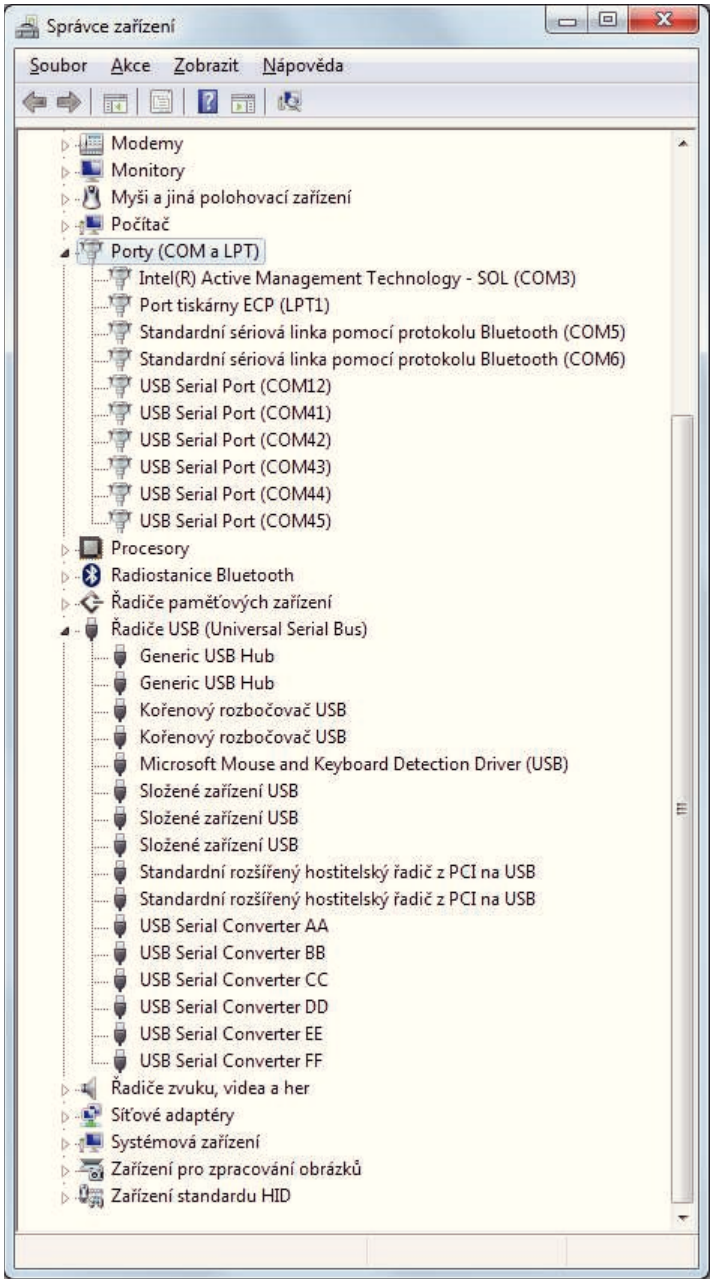
Po všech výše uvedených krocích se konečně ve *Správci zařízení* pod záložkou *Porty (COM a LPT)* objeví 6 nových virtuálních COM portů. Správně enumerované zařízení emulátoru se 6-ti porty je zobrazeno na obrázku 18 „*Správce zařízení a 6 komunikačních kanálů/portů, zdroj: autor*“ na straně 98.

Zjištění 7.2 Prakticky jsme tedy ověřili, že má smysl přidávat do ovladače více portů pro dvouportový převodník FT2232D. I přes to, že je obvod FT2232D hardwarově dvouportový, úpravou ovladače a při použití emulátoru s vhodně navrženým firmwarem, lze pomocí FTDI ovladačů obsluhovat i více portů/kanálů najednou než obvodem FT2232D podporovaných dvou. Toto zajímavé zjištění jsme demonstrovali vytvořením 6-ti kanálového emulátoru v MPLAB® X projektu „USB Device Serial Emulator FTDI 6 PORT ENUM“.

Protože všechny výše popsané kroky, které nás vedly k vytvoření 6-ti portového emulátoru byly založeny na firmware jednoportového emulátoru, je jeden kanál funkční a pracující jako převodník USB-RS232, což bylo prověřeno otestováním v programu terminálu. Využití ostatních kanálů/portů ve firmware je nyní zcela v rukách programátora.

Nakonec lze tedy s úspěchem konstatovat, že lze celkem snadno vytvořit víceportový emulátor, dokonce s větším počtem portů, než mají současné obvody FTDI.

Poznámka 7.2 Realizovaný 6-ti kanálový emulátor byl testován pouze pod operačním systémem Windows 7.



Obrázek 18: Správce zařízení a 6 komunikačních kanálů/portů, zdroj: autor

7.3 Počet portů

Technicky je možno emulátorem dosáhnout daleko více portů než 6-ti. Horní hranice počtu je však omezena technologicky. Jeden kanál využívá celkem 2 endpointy. Jeden je pro odchozí data z USB do emulátoru a druhý pro příchozí data z emulátoru zpět do USB. Protože specifikace High Speed USB 2.0 i použitý mikroprocesor PIC18F46J50 podporují maximálně 16 obousměrných endpointů, přičemž dva jsou vždy vyhrazeny pro řídící endpoint EP0, lze v emulátoru teoreticky realizovat celkem $(32 - 2)/2 = 15$ kanálů.

Tento maximální počet může být ještě omezen možnostmi MCHFSUSB frameworku, například velikostí výsledného kódu po překladu, šikovností programátora, protože při vyšším počtu kanálů roste náročnost na obsluhu USB frameworku, která musí být v intervalech do max 1.8ms a nakonec velikostí programové paměti RAM a ROM použitého mikroprocesoru.

Shrnuto, teoreticky je možno emulátorem dosáhnout maximálního počtu 15-ti souběžných (paralelních) kanálů.

7.4 Závěr více převodníků

V této kapitole jsme si představili celkem dvě možnosti realizace více převodníků jedním emulátorem. První možnost se týkala plného emulování FTDI obvodu, který má výrobcem integrováno více hardwarových portů, například dvouportový FT2232D nebo čtyřportový FT4232H. Na tomto způsobu byl plně realizován a otestován dvouportový emulátor, jehož implementace byla popsána v kapitole 6 „*USB implementace dvojitý převodník*“ na straně 87.

Druhá možnost se týkala přizpůsobení ovladače pro dvouportový FTDI obvod FT2232D přidáním dalších portů, které je schopen ovladač obsloužit. Touto cestou byl fyzicky realizován 6-ti portový emulátor. Lze tedy prohlásit, že úkol „Zvažte možnost implementace dvou a více převodníků jedním mikropočítačem“ v části více převodníků byl vyřešen.

Tímto bych na závěr obě prověřené možnosti považoval za obecně možné postupy pro tvorbu víceportového emulátoru na základě FTDI ovladačů.

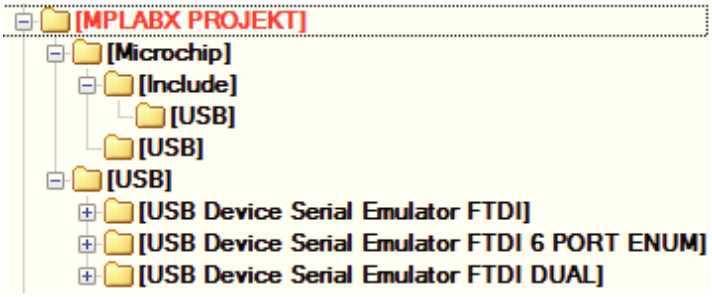
8 Projekty firmware

Součástí práce jsou tři projekty vytvořené v prostředí MPLAB® X, které obsahují veškeré zdrojové kódy, včetně přiložených souborů MCHPFSUSB frameworku, potřebných pro úspěšnou kompilaci těchto projektů.

Vytvořené projekty jsou:

- USB Device Serial Emulator FTDI
- USB Device Serial Emulator FTDI 6 PORT ENUM
- USB Device Serial Emulator FTDI DUAL

Hierarchická struktura nadřazeného adresáře *MPLABX PROJEKT* a jeho podadresářů je zobrazena na obrázku 19 „*Struktura MPLAB® X projektů, zdroj: autor*“ na straně 101.



Obrázek 19: Struktura MPLAB® X projektů, zdroj: autor

9 Závěr

Na této práci bylo demonstrováno využití mikroprocesoru PIC18F46J50 a jeho integrovaného USB a RS232 modulu k softwarové emulaci originálního FTDI převodníku USB-RS232 osazeného obvodem FT232R a využití ovladačů výrobce FTDI pro komunikaci s emulátorem.

Stanovený úkol „Programová implementace USB-RS232 převodníku v procesoru Microchip“ byl úspěšně vyřešen. Na jeho základě byl následně vyřešen také druhý stanovený úkol „Zvažte možnost implementace dvou a více převodníků jedním mikropočítačem“ a to implementací plně funkčního dvouportového převodníku a implementací 6-ti kanálového emulátoru.

U dvouportového převodníku byl jeden port realizován jako hardwarový převodník USB-RS232 a druhý port byl demonstrován jako převodník USB-RS232 se softwarovým loopbackem. Účelem 6-ti kanálového emulátoru bylo zejména demonstrovat schopnost realizace teoretické myšlenky vytvoření víceportového/vícekanálového převodníku a jeho úspěšné rozpoznání operačním systémem. Proto byl ve firmware emulátoru fyzicky realizován a otestován pouze jeden port pracující jako hardwarový převodník USB-RS232 a ostatní kanály zůstaly neimplementovány. Jejich implementace čeká pouze na šikovného programátora a jeho tvůrčí představy.

Protože sběrnice USB je stále oblíbenějším prvkem propojení externích zařízení s osobním počítačem, pevně věřím, že informace poskytnuté v této práci naleznou své uplatnění jak na katedře FEI, tak také stejně dobře poslouží i dalším vážným zájemcům z oblasti elektroniky nebo počítačové techniky.

Mým nejvyšším přínosem k této práci pak byl můj čas, který jsem věnoval pečlivému zpracování tohoto tématu, s nadějí pro budoucí využití emulátoru v praxi.

autor:

Bc. Marek Darmo, DiS.

10 Sazba textu

Text diplomové práce byl vysázen v LaTeXu pomocí třídy dokumentů `diploma` ve verzi 2.2 ze dne 16. února 2011.

Autorem třídy `diploma` je doc. Mgr. Jiří Dvorský, Ph.D., který v současnosti působí na Katedře informatiky, Fakulty elektrotechniky a informatiky, VŠB - TUO. Třída `diploma` je autorem poskytnuta k volnému použití.

Třídu `diploma` doplňují vlastní LaTeX funkce, které jsem vytvořil a použil pro úpravu sazby obrázků *InsertFigureForce*, pro sazbu referencí uvnitř textu *MyNameRef* a pro sazbu zjištění *MyFinding*.

K překladu jsem použil distribuci „TeX Live 2010“ na platformě Windows 7. Zdrojový soubor čítá téměř 3000 řádků textu a přeložený obsahuje 129 číslovaných stran. Přeložený text je dostupný v post draft formátu „PDF“ v jednostranné i oboustranné sazbě.

11 Reference

[1] *URL: <http://www.microchip.com/>*, Microchip Technology Inc., Corporate Headquarters Microchip Technology Inc. 2355 West Chandler Blvd. Chandler, Arizona, USA, 2012.

[2] *URL: <http://www.ftdichip.com/>*, Future Technology Devices International Limited, Seaward Place, Centurion Business Park Glasgow, United Kingdom, 2012.

[3] Copyright © 2006 Future Technology Devices International Ltd., *FT232R USB UART IC, Document No.: FT_000053*, FTDI pdf datasheet, Version 2.09, URL: <http://www.ftdichip.com>, 2010.

[4] Copyright © 2006 Future Technology Devices International Ltd., *AN232B-04 Data Throughput, Latency and Handshaking*, FTDI pdf datasheet, URL: <http://www.ftdichip.com>, 2006.

[5] Copyright © 2006 Future Technology Devices International Ltd., *AN232B-05 Configuring FT232R, FT2232 and FT232B Baud Rates*, FTDI pdf datasheet, URL: <http://www.ftdichip.com>, 2006.

[6] ©2011 Microchip Technology Inc., *PIC18F46J50 Data Sheet, DS39931D*, pdf datasheet, Last Updated: 04/05/2011.

[7] ©2005 Microchip Technology Inc., *MPLAB® C18 C COMPILER LIBRARIES, DS51297F*, pdf datasheet, Last Updated: 2005.

[8] Bud Caldwell, ©2008 Microchip Technology Inc., *AN1164, USB CDC Class on an Embedded Device, DS01164A*, pdf application note, Last Updated: 2008.

[9] ©2008 Microchip Technology Inc., *Microchip USB Device Firmware Framework User's Guide, DS51679B*, pdf datasheet, Last Updated: 2008.

[10] Copyright ©2010 Microchip Technology, Inc. *MCHPFSUSB Library Help*, pdf datasheet, Version v2.9c, Last Updated: 2011.

[11] Copyright © 2007, USB Implementers Forum, Inc., *Universal Serial Bus Class Definitions for Communications Devices*, Revision 1.2, Last Updated: November 16, 2007.

[12] Copyright © 2000, Compaq Computer Corporation, Hewlett-Packard Company, Intel Corporation, Lucent Technologies Inc, Microsoft Corporation, NEC Corporation, Koninklijke Philips Electronics N.V., *Universal Serial Bus Specification*, Revision 2.0, Last Updated: April 27, 2000.

[13] *URL: <http://www.ftdichip.com/Products/ICs.htm/>*, FTDI range of USB connectivity ICs, 2012.

[14] ©2010 Microchip Technology Inc., *PICkit™ 3 Programmer/Debugger User's Guide, DS51795B*, pdf datasheet, Last Updated: 7/12/2010.

[15] ©2010 Microchip Technology Inc., *PIC18F4XJ5X Full-Speed USB Demonstration Board User's Guide, DS51806B*, pdf datasheet, Last Updated: 4/9/2010.

[16] ©2004 Microchip Technology Inc., Eric Somerville., *PICmicro©Device Programming: What You Always Wanted to Know (But Didn't Know Who to Ask), DS00910A*, pdf datasheet, Last Updated: 11/23/05.

[17] DIGILENT®, URL: <http://www.digilentinc.com/> *Digilent PmodRS232™ Converter Module Board Reference Manual, Doc: 502-068*, pdf datasheet, Note: This document applies to REV B of the board., Last Updated: February 21, 2007.

[18] Copyright ©SysNucleus. *USBTrace* [počítačový program]. Ver. 2.7.0.76 ITES Habitat Centre, JNI Stadium Complex, India 2012. URL: <http://www.sysnucleus.com/> 15 Days Evaluation Version.

[19] PremimCord, URL: <http://http://www.premiumcord.com/> *USB TO RS232 CABLE, KU2-232*, BARCODE 8/592220001698, CE.

[20] Alihusain Y. Sirohiwala, Worcester Polytechnic Institute, *An Interactive Application Demo for the AD8295*, pdf MQP_Final_Report_IC_APP_2008, Last Updated:2008.

[21] *URL: <http://www.intra2net.com/en/developer/libftdi/index.php>*, libFTDI - FTDI USB driver with bitbang mode, Version 0.20 Last Updated: 2012-03-19.

[22] *URL: <http://yosemitefoothills.com/Electronics/>*, Linux Communication with FT2232C based USB Devices, Last Updated: February 7, 2009.

[23] *URL: <http://lxr.free-electrons.com/source/drivers/usb/serial/ftdi_sio.h>*, USB FTDI SIO driver, header file, Version: 3.3, Last Updated: 2011.

[24] *URL: <http://lxr.free-electrons.com/source/drivers/usb/serial/ftdi_sio.c>*, USB FTDI SIO driver, source file, Version: 3.3, Last Updated: 2011.

[25] *URL: <http://ftdi-usb-sio.sourceforge.net/>*, FTDI SIO, 8U232AM and 245 Linux USB serial driver, Last Updated: February 2002.

[26] *URL: <http://www.google.com/>*, Google internet browser, English version, 2012.

[27] *URL: <http://www.beyondlogic.org/usbnutshell/usb1.shtml/>*, 2010 ©USB in a NutShell, Last Updated: Friday, 17-Sep-2010.

A Výpisy kódu

```
/* Device Descriptor */
ROM USB_DEVICE_DESCRIPTOR device_dsc=
{
    0x12,                   // sizeof(USB_DEVICE_DESCRIPTOR)
    USB_DESCRIPTOR_DEVICE, // DEVICE descriptor type
    0x0200,                 // USB Spec Release Number in BCD format
    0x00,                   // Class Code
    0x00,                   // Subclass code
    0x00,                   // Protocol code
    USB_EP0_BUFF_SIZE,  // Max packet size for EP0, see usb_config.h
    0x0403,                 // Vendor ID
    0x6001,                 // Product ID
    0x0600,                 // Device release number in BCD format
    0x01,                   // Manufacturer string index
    0x02,                   // Product string index
    0x03,                   // Device serial number string index
    0x01                    // Number of possible configurations
};

/* Configuration 1 Descriptor */
ROM BYTE configDescriptor1[]={
    /* Configuration Descriptor */
    0x09,                   //sizeof(USB_CFG_DSC)
    USB_DESCRIPTOR_CONFIGURATION, // CONFIGURATION descriptor type
    32,0,                   // Total length of data for this cfg
    1,                      // Number of interfaces in this cfg
    1,                      // Index value of this configuration
    0,                      // Configuration string index
    _DEFAULT,               // Bus Powered Attributes, see usb_device.h
    50,                     // Max power consumption (x2 mA)

    /* Interface Descriptor */
    0x09,                   //sizeof(USB_INTF_DSC),
    USB_DESCRIPTOR_INTERFACE, // INTERFACE descriptor type
    0,                      // Interface Number
    0,                      // Alternate Setting Number
    2,                      // Number of endpoints in this intf
    0xFF,                   // Class code
    0xFF,                   // Subclass code
    0xFF,                   // Protocol code
    2,                      // Interface string index

    /* Endpoint Descriptor */
    0x07,                   //sizeof(USB_EP_DSC)
    USB_DESCRIPTOR_ENDPOINT,//Endpoint Descriptor
    _EP01_IN,               //EndpointAddress
    _BULK,                  // Attributes
    0x40,0x00,              //size
    0x00,                   // Interval

    /* Endpoint Descriptor */
    0x07,                   //sizeof(USB_EP_DSC)
    USB_DESCRIPTOR_ENDPOINT,//Endpoint Descriptor
```

```
    _EP02_OUT,              //EndpointAddress
    _BULK,                  // Attributes
    0x40,0x00,              //size
    0x00                    // Interval
};

//Language code string descriptor US English (0x0409)
ROM struct{BYTE bLength;BYTE bDscType;WORD string[1];}sd000={
sizeof(sd000),USB_DESCRIPTOR_STRING,{0x0409}};

//Manufacturer string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD string[4];}sd001={
sizeof(sd001),USB_DESCRIPTOR_STRING,
{'F','T','D','I'}
};

//Product string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD string[20];}sd002={
sizeof(sd002),USB_DESCRIPTOR_STRING,
{'U','S','B',' ','<','-','>',' ','S','e','r','i','a','l',' ',
'C','a','b','l','e'}
};

//Serial number string descriptor
ROM struct{BYTE bLength;BYTE bDscType;WORD string[8];}sd003={
sizeof(sd003),USB_DESCRIPTOR_STRING,
{'F','T','R','6','9','S','3','6'}
};

//Array of configuration descriptors
ROM BYTE *ROM USB_CD_Ptr[]=
{
    (ROM BYTE *ROM)&configDescriptor1
};

//Array of string descriptors
ROM BYTE *ROM USB_SD_Ptr[USB_NUM_STRING_DESCRIPTORS]=
{
    (ROM BYTE *ROM)&sd000,
    (ROM BYTE *ROM)&sd001,
    (ROM BYTE *ROM)&sd002,
    (ROM BYTE *ROM)&sd003
};
```

Výpis 24: Deskriptor emulátoru, zdroj: usb_descriptors_ftdi.c

B Tabulky

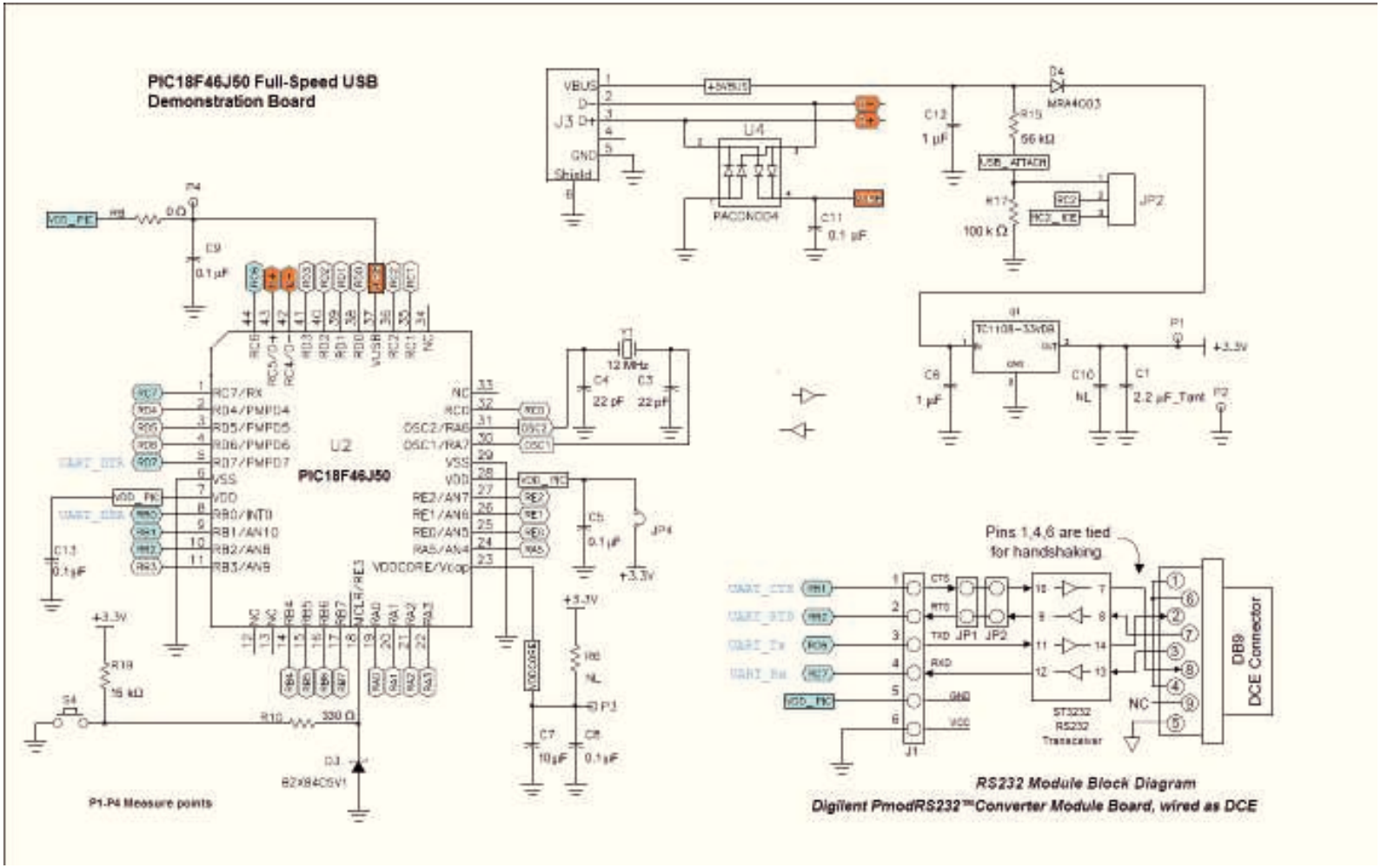
Test emulátoru - konfigurace RS232 port loopback								
Rychlost	Znaky - spolehlivost	Text				Software/Hardware Handshaking		
baudrate	Odeslání/Příjem	Manuální	Krátký	Dlouhý	Velmi dlouhý	none	SW XON/XOFF	HW RTS/CTS
300	pass	pass	pass	pass	pass	pass	pass	pass
1200	pass	pass	pass	pass	pass	pass	pass	pass
2400	pass	pass	pass	pass	pass	pass	pass	pass
4800	pass	pass	pass	pass	pass	pass	pass	pass
9600	pass	pass	pass	pass	pass	pass	pass	pass
19200	pass	pass	pass	pass	pass	pass	pass	pass
38400	pass	pass	pass	pass	pass	pass	pass	pass
57600	pass	pass	pass	pass	pass	pass	pass	pass
11500	pass	pass	pass	pass	pass	pass	pass	pass

Tabulka 15: Test emulátoru - konfigurace RS232 port loopback, zdroj: autor

Test emulátoru - konfigurace loopback emulátor a originální převodník FTDI								
Rychlost	Znaky - spolehlivost	Text				Software/Hardware Handshaking		
baudrate	Odeslání/Příjem	Manuální	Krátký	Dlouhý	Velmi dlouhý	none	SW XON/XOFF	HW RTS/CTS
300	pass	pass	pass	pass	pass	pass	pass	pass
1200	pass	pass	pass	pass	pass	pass	pass	pass
2400	pass	pass	pass	pass	pass	pass	pass	pass
4800	pass	pass	pass	pass	pass	pass	pass	pass
9600	pass	pass	pass	pass	pass	pass	pass	pass
19200	pass	pass	pass	pass	pass	pass	pass	pass
38400	pass	pass	pass	pass	pass	pass	pass	pass
57600	pass	pass	pass	pass	pass	pass	pass	pass
11500	pass	pass	pass	pass	pass	pass	pass	pass

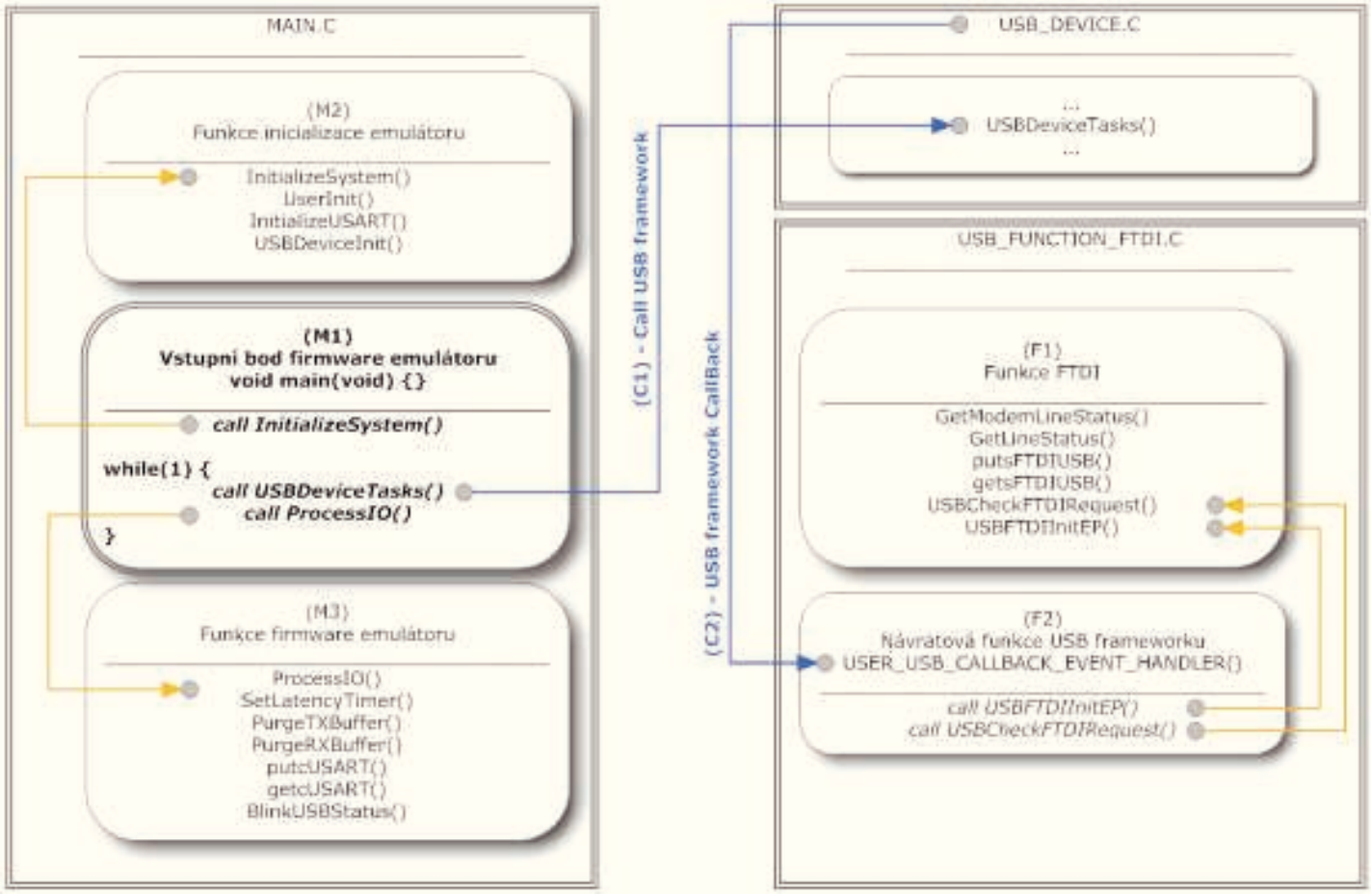
Tabulka 16: Test emulátoru - konfigurace loopback přes originální FTDI převodník, zdroj: autor

C Schéma

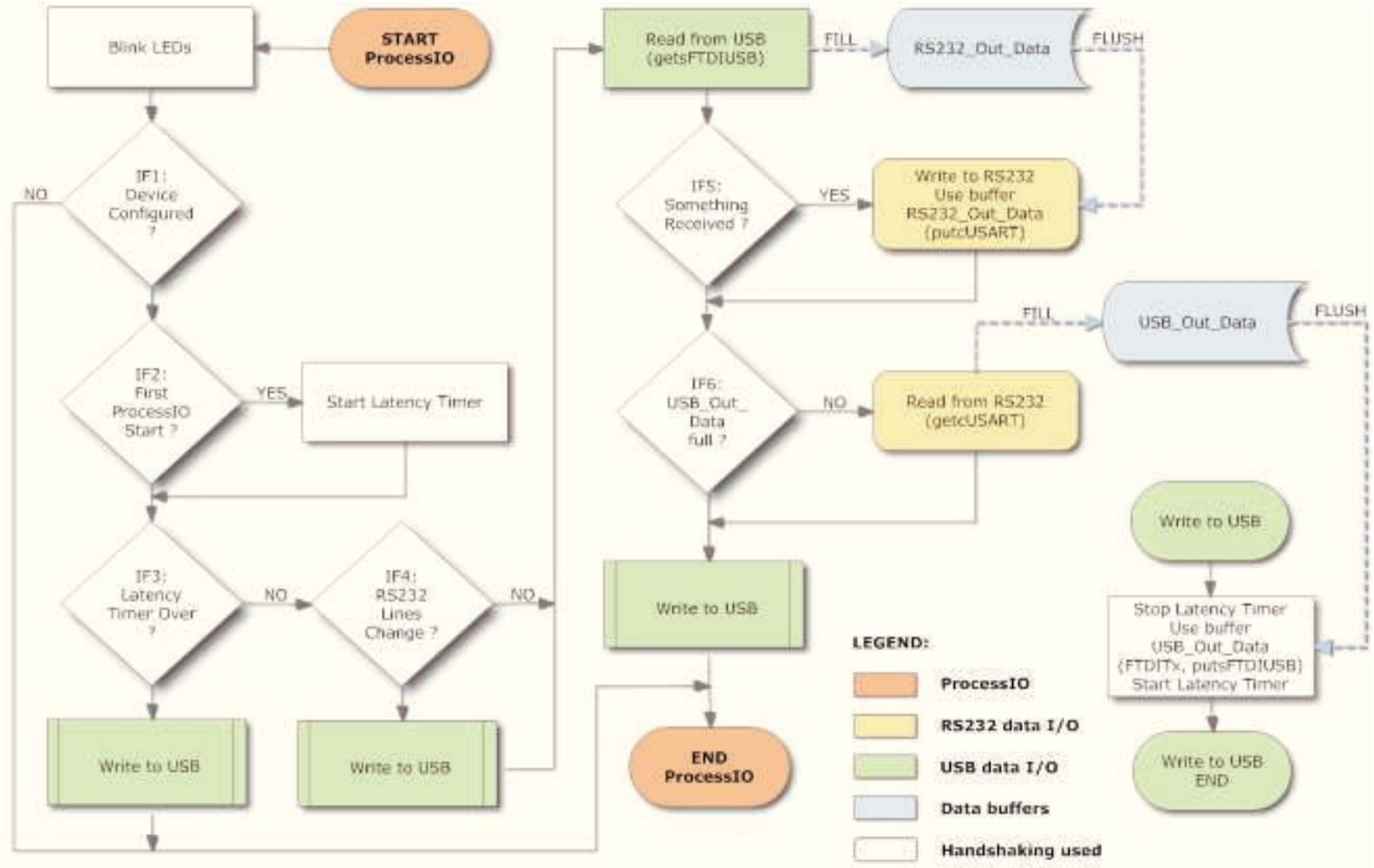


Obrázek 20: Schéma zapojení emulátoru FT232R, zdroj: autor

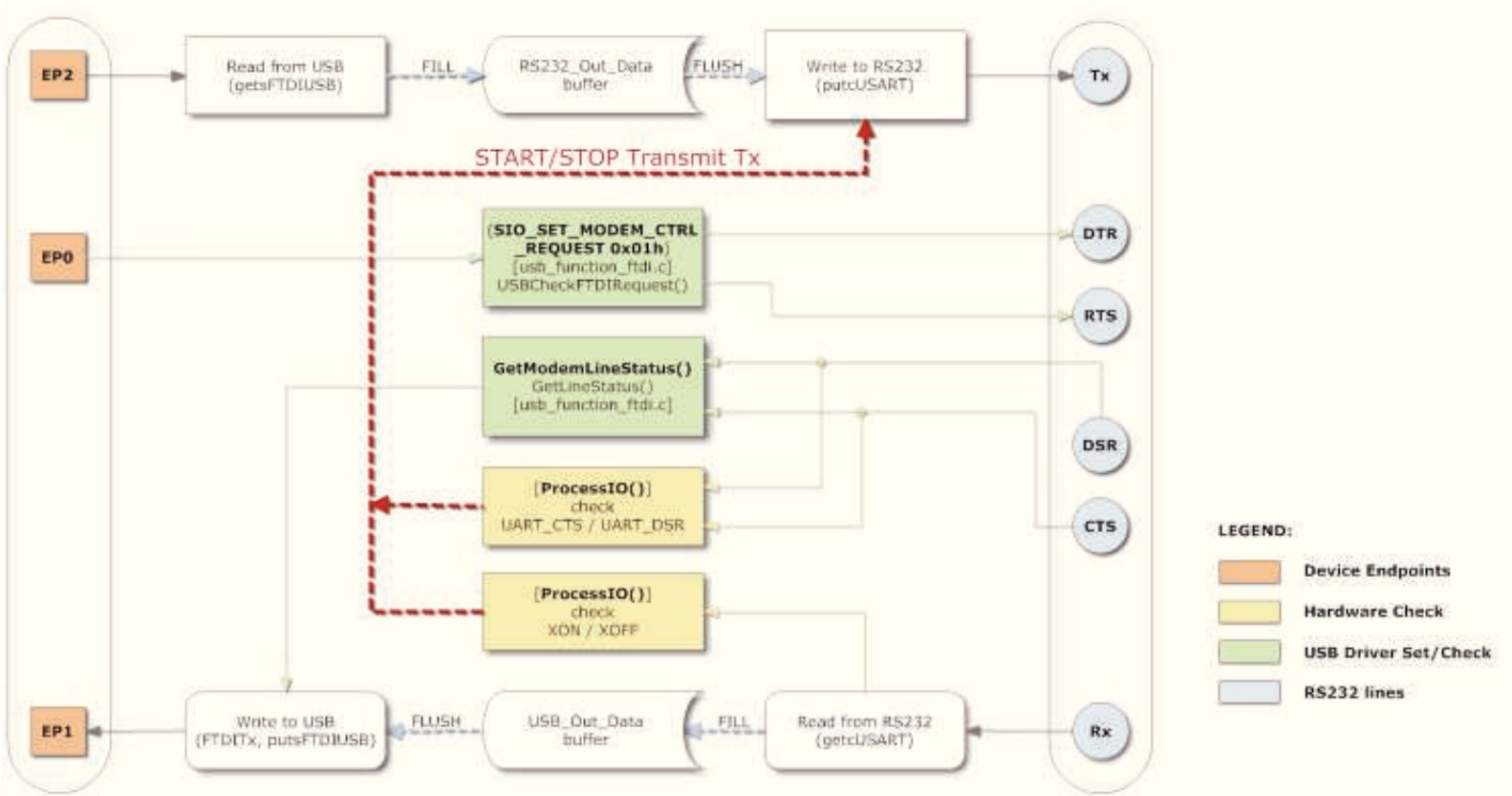
D Vývojové diagramy



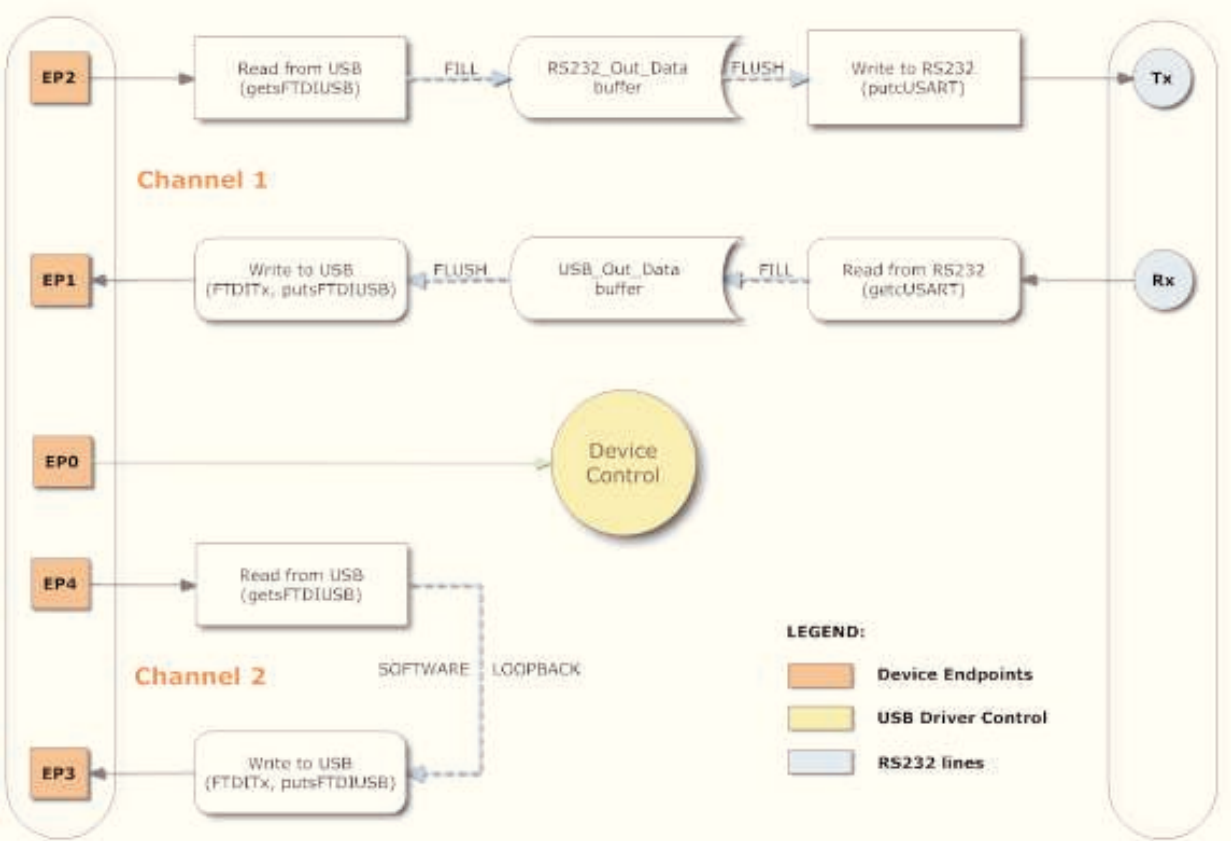
Obrázek 21: Blokový diagram firmware emulátoru, zdroj: autor



Obrázek 22: Vývojový diagram funkce ProcessIO firmware emulátoru, zdroj: autor

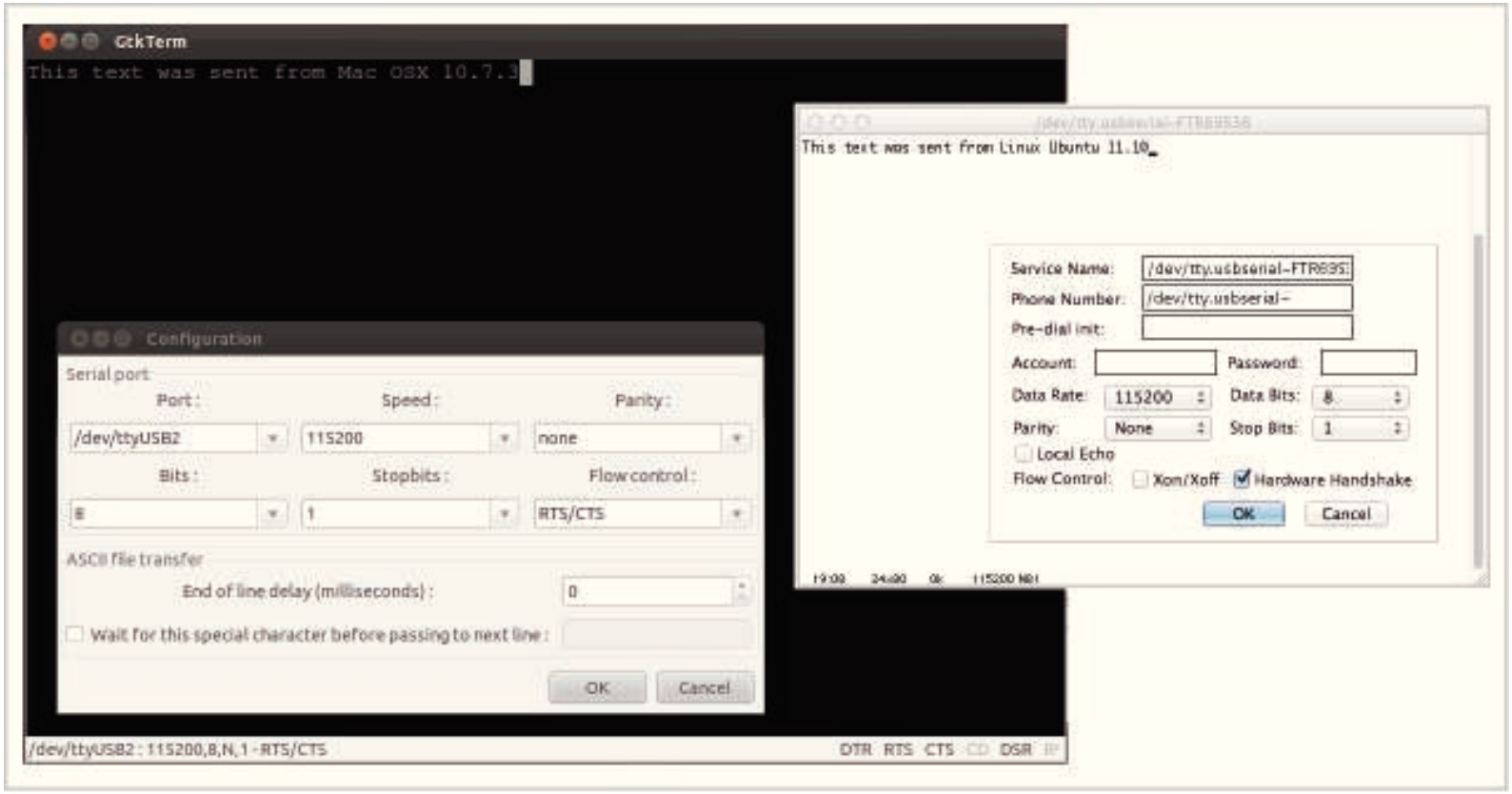


Obrázek 23: Vývojový diagram handshakingu emulátoru, zdroj: autor



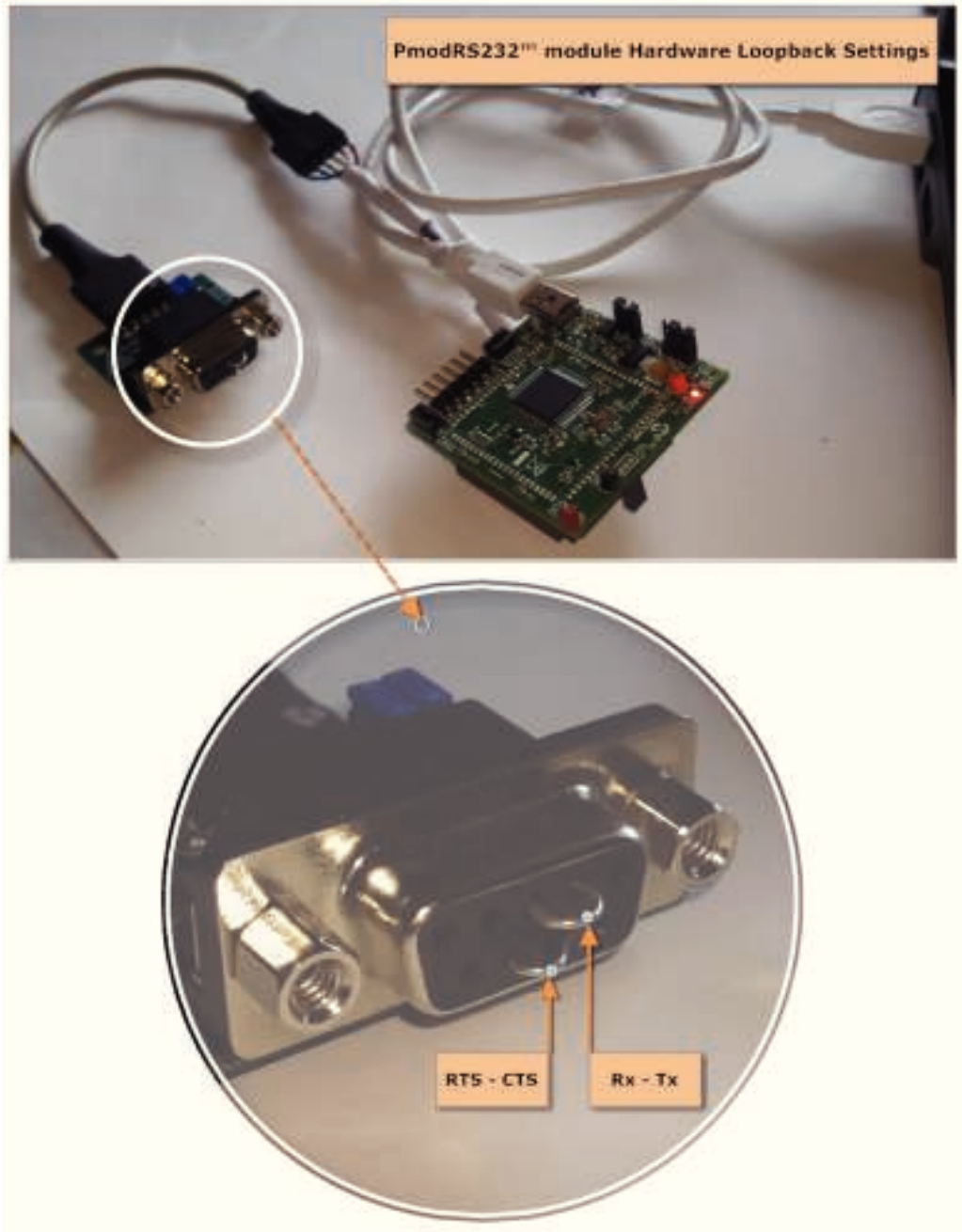
Obrázek 24: Realizace kanálů dvojího převodníku, zdroj: autor

E Screenshoty

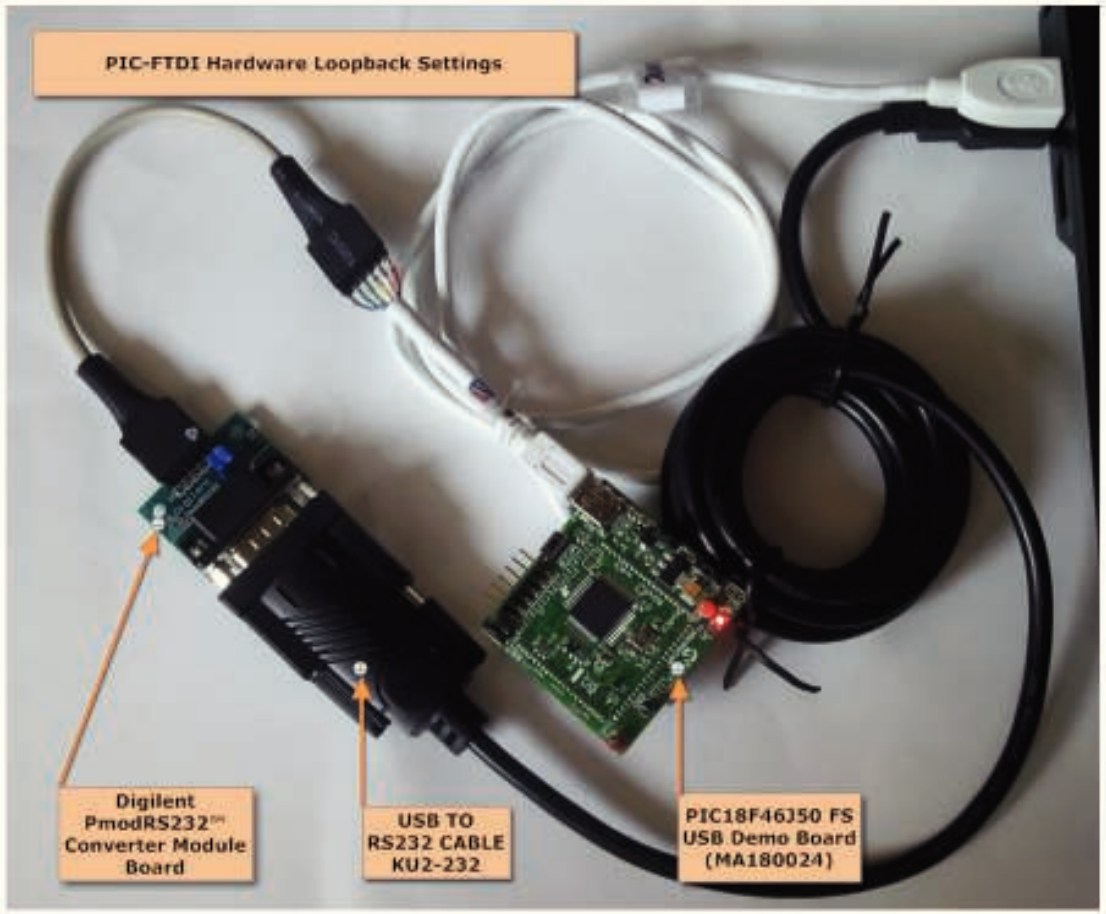


Obrázek 25: Obsah terminálů mezipropojení OS Mac OS X a Linux Ubuntu, zdroj: autor

F Fotografie zařízení



Obrázek 26: Hardware emulátoru - RS232 port loopback, zdroj: autor



Obrázek 27: Hardware emulátoru - loopback přes originální FTDI převodník, zdroj: autor